Урок 2 - Базовая структура приложения

Оглавление

Цель урока

1. Разбираем макет приложения
2. Строим структуру интерфейса главного окна - обзор визуальных элементов управления
3. Рассматриваем основные структурные модули логики

4. Базовые понятия шаблона MVVM
5. Привязка к данным

На прошлом занятии мы создали заготовку проекта, опробовали работу главной бизнес-функции и создали приложение-прототип с WPF-интерфейсом пользователя. Займёмся развитием основного проекта и создадим основу интерфейса - его макет и основные элементы бизнес-логики.

Интерфейс пользователя

Займёмся интерфейсом. Нем необходимо создать макет разметки главного окна.

На текущий момент разметка главного окна выглядит следующим образом:

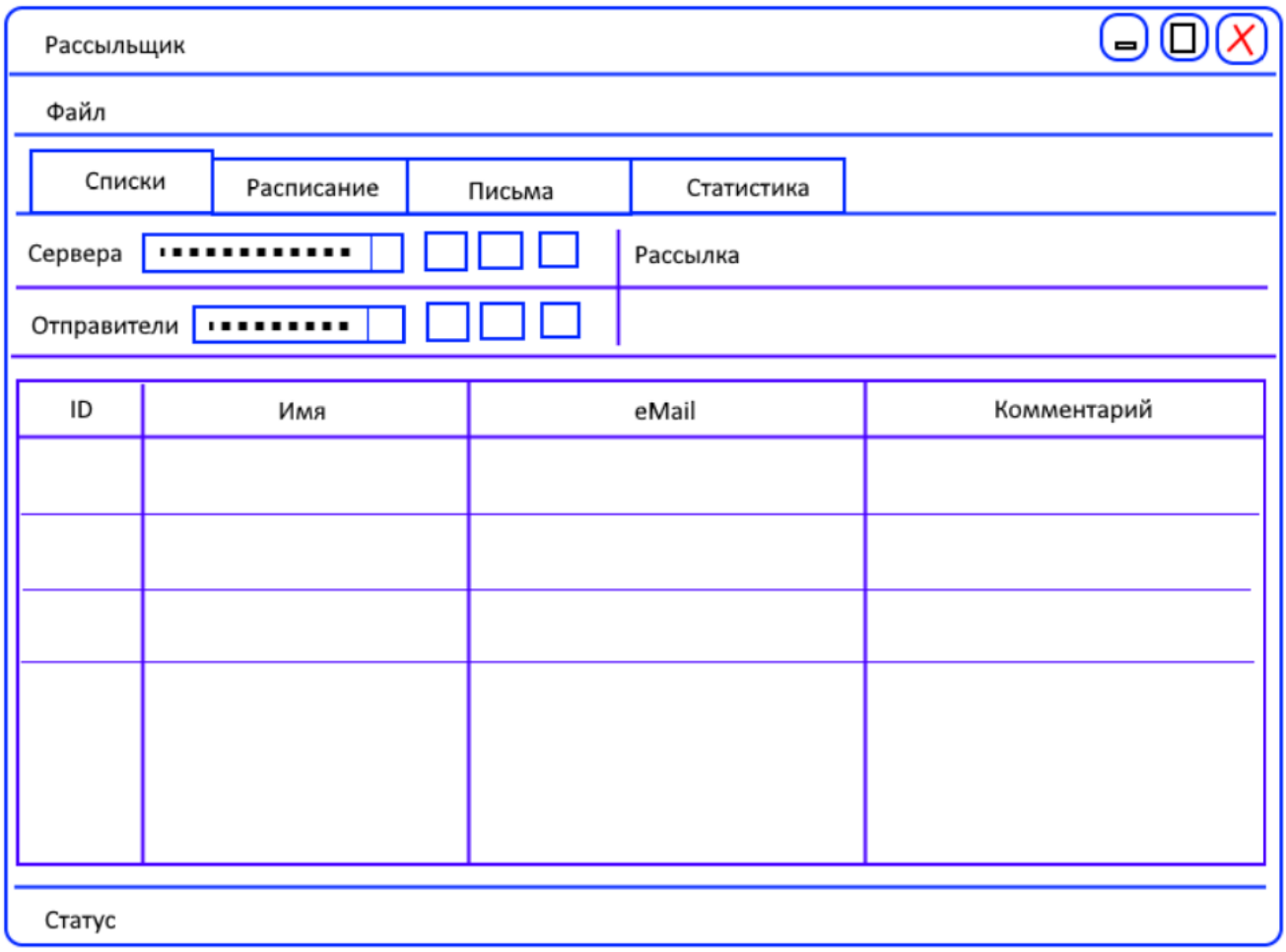
```
<Window x:Class="MailSender.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:MailSender"
        mc:Ignorable="d"
        Title="MainWindow" Height="274" Width="476">
    <Grid>

    </Grid>
</Window>
```

Окно `Window` представляет собой визуальный элемент-контейнер для содержимого. Внутри него может находится лишь один дочерний визуальный объект. Если положить внутри окна кнопку...

```
<Window x:Class="MailSender.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:MailSender"
        mc:Ignorable="d"
        Title="MainWindow" Height="274" Width="476">
    <Button>
        Кнопка
    </Button>
</Window>
```

... то мы получим окно с единственной кнопкой. И больше внутри окна мы разместить ничего не сможем. Если нам окно с одной кнопкой не нужно, то внутрь окна надо разместить контейнер, способный вместить в себя набор из нужного нам количества других элементов.

По умолчанию Студия создаёт нам шаблон окна в котором находится контейнер `Grid`. Мы заменим его на `DockPanel`. Это упростит нам создание макета, описанного на прошлом занятии



Главное меню

В верхней части окна по задумке должно быть расположено главное меню. В нижней - статусная панель. В центральной части - панель с вкладками. `DockPanel` позволяет "притягивать" элементы к краям панели и растягивать последний добавленный в панель элемент на всю оставшуюся поверхность.

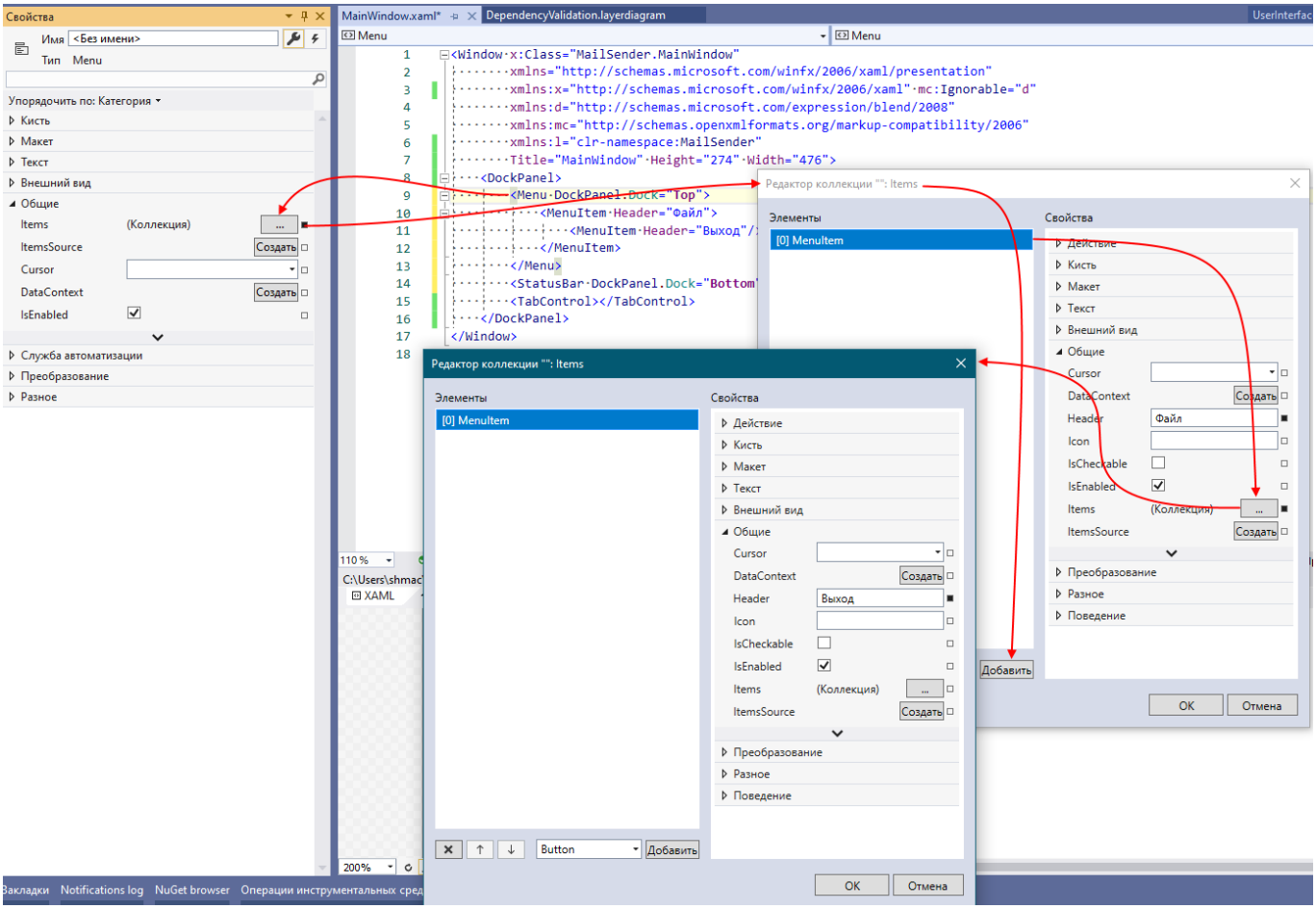
```
<Window x:Class="MailSender.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" mc:Ignorable="d"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:l="clr-namespace:MailSender"
        Title="MainWindow" Height="274" Width="476">
    <DockPanel>
        <Menu DockPanel.Dock="Top"></Menu>
        <StatusBar DockPanel.Dock="Bottom"></StatusBar>
        <TabControl></TabControl>
    </DockPanel>
</Window>
```

Разместим в док-панели меню, статус-бар и таб-контрол. Меню "прижмём" к верхнему краю док-панели, статус-бар - к нижнему краю, а таб-контрол займём всё оставшееся промежуточное пространство.

Добавим в меню пункт "Файл/Выход".

```
<Menu DockPanel.Dock="Top">
    <MenuItem Header="Файл">
        <MenuItem Header="Выход"/>
    </MenuItem>
</Menu>
```

Сам элемент меню можно добавить в окно и в визуальном дизайнере мышкой перетащив его из панели элементов. А структуру меню можно настроить также в дизайнере в панели Свойств.



Использование визуального дизайнера оправдано на начальных этапах обучения, либо как подсказка о доступных возможностях разметки (дизайнер за нас сформирует разметку, которую мы сможем изучить).

Элемент `Menu` представляет собой контейнер для отображения вложенных в него элементов меню (`MenuItem`). По своей сути меню представляет собой древовидный элемент визуализации списочного иерархического содержимого. Аналогичен `TreeView` - дереву элементов. Каждый элемент меню `MenuItem` также представляет собой меню снабжённое заголовком (свойство `Header` отвечает за указание текста, который будет выведен на экран). Внутрь элемента `MenuItem` можно добавить любое количество воженных `MenuItem`, а также элемент-разделитель `<Separator/>` рисующий горизонтальную линию.

Панель статуса

Статусная панель размещается в нижней части экрана и отображает текущее состояние программы

Панель статуса состоит из элементов-панели-статуса (`StatusBarItem`) каждый из которых представляет собой контейнер для размещения любых других элементов.

Разместим в статусной панели два таких элемента: в первом разместим элемент масштабирования окна (`ResizeGrip`), а во втором простейший элемент визуализации текста `TextBlock`

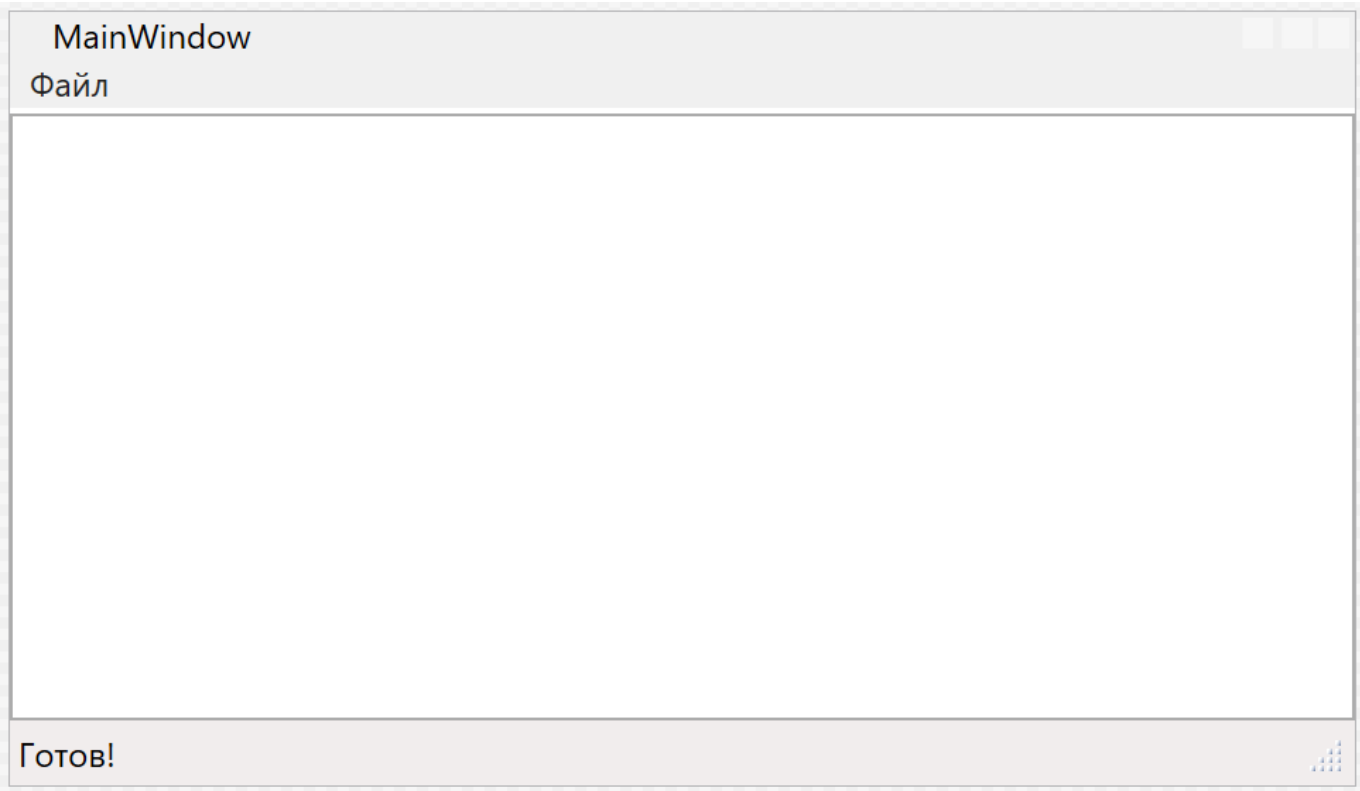
```
<StatusBar DockPanel.Dock="Bottom">
    <StatusBarItem>
        <ResizeGrip/>
    </StatusBarItem>
    <StatusBarItem>
        <TextBlock Text="Готов!"/>
    </StatusBarItem>
</StatusBar>
```

При таком размещении элемент `ResizeGrip` расположен слева, а текст - следом за ним.

Для переноса `ResizeGrip` в правый угол изменим расстановку `StatusBarItem`. `StatusBar` для визуализации своих дочерних элементов использует внутри `DockPanel`. Поэтому мы можем указать каждому элементу `StatusBarItem` место в `DockPanel`, которое нам нужно.

```
<StatusBar DockPanel.Dock="Bottom">
    <StatusBarItem DockPanel.Dock="Right">
        <ResizeGrip/>
    </StatusBarItem>
    <StatusBarItem>
```

```
        <TextBlock Text="Готов!"/>
    </StatusBarItem>
</StatusBar>
```



Теперь элемент занял нужное место в статусной строке.

Системные элементы управления заняли своё законное место, и разметка окна на текущий момент выглядит так:

```
<Window x:Class="MailSender.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" mc:Ignorable="d"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:l="clr-namespace:MailSender"
        Title="MainWindow" Height="274" Width="476">
    <DockPanel>
        <Menu DockPanel.Dock="Top">
            <MenuItem Header="Файл">
                <MenuItem Header="Выход"/>
            </MenuItem>
        </Menu>

        <StatusBar DockPanel.Dock="Bottom">
            <StatusBarItem DockPanel.Dock="Right">
                <ResizeGrip/>
            </StatusBarItem>
            <StatusBarItem>
                <TextBlock Text="Готов!"/>
            </StatusBarItem>
```

```
        </StatusBar>

        <TabControl>
            <!-- Основное содержимое вкладок должно быть здесь -->
        </TabControl>
    </DockPanel>
</Window>
```

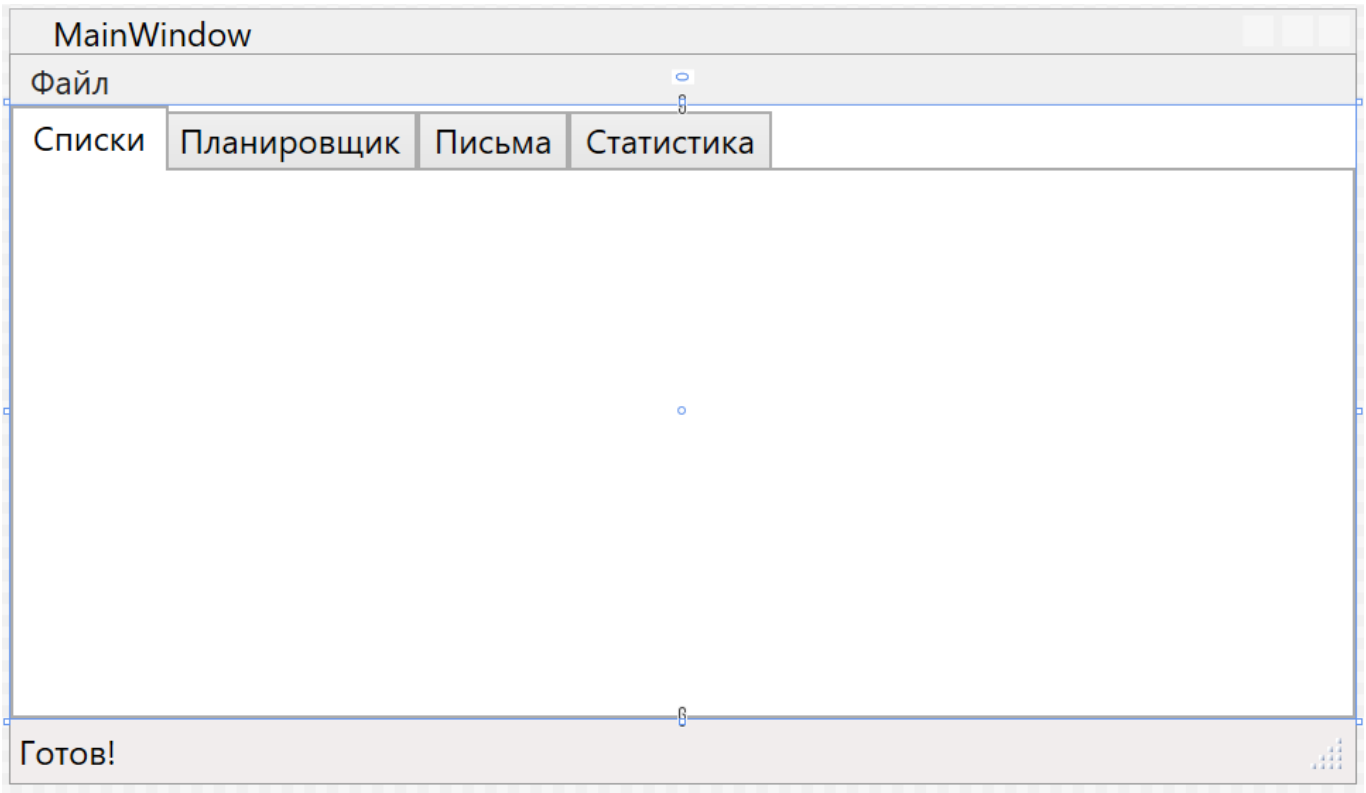
Разметку в окне имеет смысл разделять пустыми строками и делать выравнивание чтобы поддерживать максимально возможный уровень читаемости кода разметки. Отделим основной контент окна от системных элементов пустой строкой.

Экран управления списками рассылки

Добавим в `TabControl` нужные нам вкладки - элементы `TabItem`.

```
<TabControl>
    <TabItem Header="Списки"></TabItem>
    <TabItem Header="Планировщик"></TabItem>
    <TabItem Header="Письма"></TabItem>
    <TabItem Header="Статистика"></TabItem>
</TabControl>
```

В результате мы получим окно следующего вида:



Элемент `TabItem` представляет собой одну вкладку `TabControl`. Вкладка `TabItem` представляет собой элемент управления содержимым (`ContentControl`) снабжённый заголовком (базовый класс `HEaderedContentControl`). У него есть свойство `Header` представляющий разметку, Выводимую в

ярлычке вкладки. Заголовок может содержать простой текст, либо может содержать более сложную разметку.

```
<TabItem Header="Списки">
    <TabItem.Header> <!-- Любое свойство можно развернуть в элемент -->
        <StackPanel Orientation="Horizontal">
            <TextBlock Text="Заголовок вкладки"/>
            <Button Foreground="Red" Margin="5,0,0,0">X</Button>
        </StackPanel>
    </TabItem.Header>
    <TextBlock Text="123" VerticalAlign="Center" HorizontalAlign="Center"/>
</TabItem>
```

В данном примере разметки в заголовок вкладки выводится текст и кнопку "закрыть" красного цвета с зазором 5 "точек" между кнопкой и текстом. в содержимое вкладки добавляется текст 123 по центру.

Таким образом, `TabItem` - элемент, имеющий заголовок и содержимое. Причём, содержимым может быть только один визуальный элемент (как и в окне). Чтобы во вкладке разместить более одного визуального элемента нам опять понадобится панель-компоновки.

Можно в качестве панели-компоновки здесь также воспользоваться `DockPanel`. Но для разнообразия здесь мы возьмём `Grid`.

Разобьём `Grid` на две строки. Высоту первой строки сделаем автоматической

```
<TabItem Header="Списки">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
        </Grid.RowDefinitions>

        <!-- Содержимое -->
    </Grid>
</TabItem>
```

Разместим здесь в первой строке таблицы лоток для панелей инструментов `ToolBarTray`, а во второй строке - элемент для отображения данных в табличной форме `DataGrid`.

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition/>
    </Grid.RowDefinitions>

    <ToolBarTray Grid.Row="0">
        <!-- Панели инструментов -->
    </ToolBarTray>
```

```
    <DataGrid Grid.Row="1" />
</Grid>
```

Панели инструментов

В лоток панели инструментов разместим сами панели:

```
<ToolBarTray Grid.Row="0">
    <ToolBar Header="Сервера"></ToolBar>
    <ToolBar Header="Отправители"></ToolBar>
    <ToolBar Header="Планировщик"></ToolBar>
</ToolBarTray>
```

Панели инструментов `ToolBar` очень похожи на "вкладки" `TabItem`. Это также элемент с заголовком, но внутри располагается набор элементов панели. Для каждой панели можно указывать как она будет расположена в лотке. Чтобы расположить панель во второй строке, достаточно установить её индекс `Band` у панели `<ToolBar Header="Отправители" Band="1" BandIndex="0"></ToolBar>` и номер панели в строке `BandIndex`.

Рассмотрим панель инструментов управления серверами. Добавим в неё элемент управления списком элементов с `ComboBox` - выпадающий список и ещё три кнопки.

```
<ToolBar Header="Сервера">
    <ComboBox MinWidth="90" MaxWidth="140" SelectedIndex="0">
        <ComboBoxItem>smtp.yandex.ru:587</ComboBoxItem>
        <ComboBoxItem>smtp.gmail.com</ComboBoxItem>
        <ComboBoxItem>smtp.mail.ru</ComboBoxItem>
    </ComboBox>

    <Button Content="Добавить"/>
    <Button Content="Удалить"/>
    <Button Content="Редактировать"/>
</ToolBar>
```

Я добавил в панель `ComboBox` и задал его ширину "мягко". Размеры визуальных элементов в WPF могут определяться контейнером, в котором они "лежат", размеры могут определяться автоматических исходя из содержимого, либо может быть задан жёстко (через задание свойств `Width` и `Height`). Но также мы можем дать некоторую "свободу выбора" визуальному элементу подстраиваться под содержимое задавая не сами размеры `Width` и `Height`, а ограничения на них сверху и снизу `MinWidth`, `MinHeight` и `MaxWidth`, `MaxHeight`.

Внутри `ComboBox` были размещены тестовые даные и указано `SelectedIndex="0"`, что выбранным является первый элемент.

Аналогично оформим панель управления отправителями. В итоге разметка панелей инструментов на данном этапе будет выглядеть следующим образом:

```
<ToolBarTray Grid.Row="0">
    <ToolBar Header="Сервера">
        <ComboBox MinWidth="90" MaxWidth="140" SelectedIndex="0">
            <ComboBoxItem>smtp.yandex.ru:587</ComboBoxItem>
            <ComboBoxItem>smtp.gmail.com</ComboBoxItem>
            <ComboBoxItem>smtp.mail.ru</ComboBoxItem>
        </ComboBox>

        <Button Content="Добавить"/>
        <Button Content="Удалить"/>
        <Button Content="Редактировать"/>
    </ToolBar>
    <ToolBar Header="Отправители" Band="1" BandIndex="0">
        <ComboBox MinWidth="90" MaxWidth="140" SelectedIndex="0">
            <ComboBoxItem>Иванов:ivanov@mail.ru</ComboBoxItem>
            <ComboBoxItem>Петров:petrov@mail.ru</ComboBoxItem>
            <ComboBoxItem>Сидоров:sidorov@mail.ru</ComboBoxItem>
        </ComboBox>

        <Button Content="Добавить"/>
        <Button Content="Удалить"/>
        <Button Content="Редактировать"/>
    </ToolBar>
    <ToolBar Header="Планировщик">
        <Button Content="Запланировать"/>
    </ToolBar>
</ToolBarTray>
```

Панель инструментов готова. Теперь доработаем таблицу получателей для начала просто добавив нужные столбцы.

```
<DataGrid Grid.Row="1">
    <DataGrid.Columns>
        <DataGridCheckBoxColumn/>
        <DataGridComboBoxColumn/>
        <DataGridHyperlinkColumn/>
        <DataGridTemplateColumn/>
        <DataGridTextColumn Header="ID"/>
        <DataGridTextColumn Header="Имя" MinWidth="120"/>
        <DataGridTextColumn Header="Адрес" MinWidth="150"/>
        <DataGridTextColumn Header="Описание" Width="*"/>
    </DataGrid.Columns>
</DataGrid>
```

У элемента управления `DataGrid` есть свойство `Columns` представляющее собой коллекцию колонок, в которую мы можем добавлять и настраивать нужные нам виды колонок. На выбор предоставляется следующий набор вариантов колонок:

- `DataGridTextColumn` - обычный текстовый столбец, предназначенный для отображения текста и числовых данных
- `DataGridCheckBoxColumn` - столбец с флажками `CheckBox`
- `DataGridComboBoxColumn` - столбец, ячейки которого представляют выпадающие списки `ComboBox`
- `DataGridHyperlinkColumn`- столбез, ячейками которого являются гиперссылки `Label`
- `DataGridTemplateColumn`- если Вас не устраивает ни один из вышеперечисленных типов столбцов, то `DataGridTemplateColumn` Ваш выбор: Вы можете самостоятельно описать шаблон ячейки этого столбца так, как Вам будет нужно.

Теперь займёмся планировщиком...

Экран управления планировщиком рассылки почты

Для организации пространства вкладки управления планировщиком также воспользуемся панелью компоновки `Grid`. Разобъём панель на две колонки. Ширину первой колонки зададим как автоматическую (пусть колонка сама определится в зависимости от того что будет в ней расположено). Я заранее знаю, что хочу разместить в ней один визуальный элемент, который имеет строго определённый размер (по крайней мере ширину), который и задаст мне размер колонки - это будет `Calendar` - компонент для выбора даты.

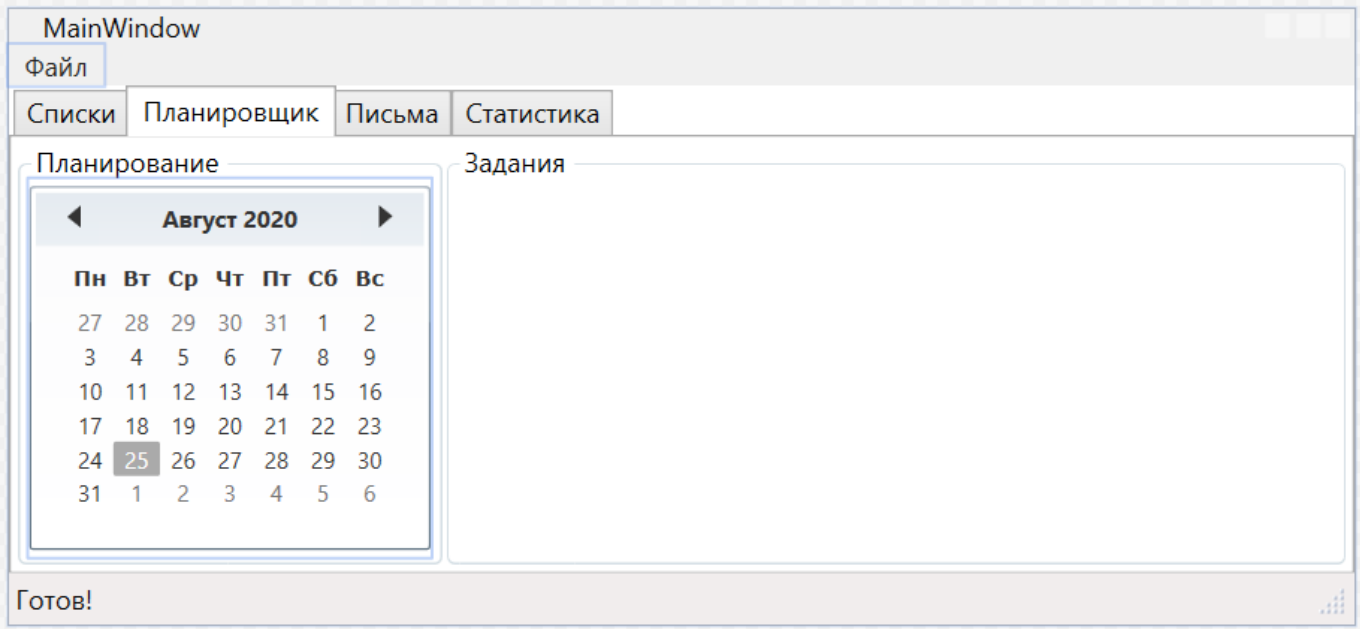
Для начала "уложим" в каждую из получившихся колонок таблицы по одному элементу `GroupBox` - элементу, позволяющему нарисовать вокруг содержимого рамку с заголовком:

```
<Grid>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>

    <GroupBox Grid.Column="0" Header="Планирование"></GroupBox>
    <GroupBox Grid.Column="1" Header="Задания"></GroupBox>
</Grid>
```

В группу с заголовком "планировщик" разместим панель `StackPanel` (размещающую своё содержимое вертикально сверху вниз друг за другом). И в панель добавим первый её элемент - `Calendar`.

```
<GroupBox Grid.Column="0" Header="Планирование">
    <StackPanel>
        <Calendar/>
    </StackPanel>
</GroupBox>
```

Внешний вид вкладки в дизайнере при этом должен быть следующим:



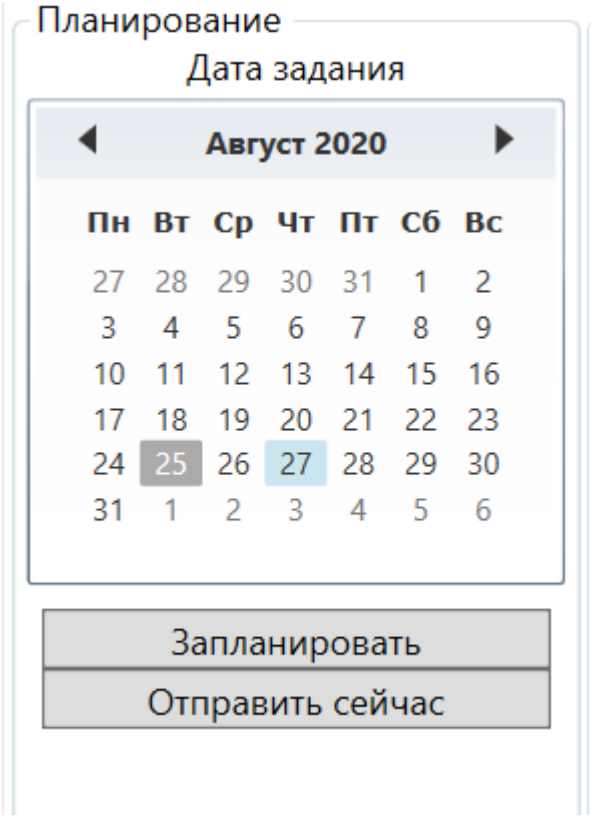
Объект `Calendar` даёт возможность отображения и выбора даты. Его основным свойством для этого является `SelectedDate`. Также данный элемент позволяет выбирать несколько дат. В этом случае набор выбранных дат можно установить/забрать из свойства `SelectedDates`. Календарь можно подписать, сверху добавив в стек-панель элемент `TextBlock` выше.

```
<GroupBox Grid.Column="0" Header="Планирование">
    <StackPanel>
        <TextBlock Text="Дата задания" HorizontalAlignment="Center"/>
        <Calendar SelectedDate="2020-08-27"/>
    </StackPanel>
</GroupBox>
```

Под календарём добавим две кнопки: "запланировать" и "отправить сейчас". Так как обе кнопки можно выделить в один логический элемент управления, их можно сгруппировать в одну панель. Используем для этого панель `UniformGrid` для того, чтобы она обеспечила равные размеры для обеих кнопок.

```
<GroupBox Grid.Column="0" Header="Планирование">
    <StackPanel>
        <TextBlock Text="Дата задания"
                   HorizontalAlignment="Center"/>
        <Calendar SelectedDate="2020-08-27"/>
        <UniformGrid Columns="1" Margin="5">
            <Button Content="Запланировать"/>
            <Button Content="Отправить сейчас"/>
        </UniformGrid>
    </StackPanel>
</GroupBox>
```

Рамка `Margin="5"` позволяет визуально отделить созданную группу кнопок от остальных визуальных элементов.



Теперь перейдём в правую часть этой вкладки - "нарисуем" в ней макет списка для отображения и управления заданиями.

Разместим в правой части вкладки элемент `ListBox` - простейший элемент для отображения списков (списочных данных).

```
<GroupBox Grid.Column="1" Header="Задания">
    <ListBox/>
</GroupBox>
```

Содержимым `ListBox` являются его элементы - `ListBoxItem`. При отображении данных обычно `ListBox` создаёт эти "контейнеры" самостоятельно "оборачивая" каждый из отображаемых элементов данных. Сейчас мы создадим их вручную и "нарисуем" их содержимое.

`ListBox` позволяет отобразить любые списочные данные в нужной визуальной форме. Данных пока у нас нет, поэтому внутреннюю его организацию создадим пока вручную чтобы увидеть весь визуальный макет в целом.

```
<GroupBox Grid.Column="1" Header="Задания">
    <ListBox>
        <ListBoxItem HorizontalContentAlignment="Stretch">
            <Border BorderBrush="Blue" BorderThickness="1"
                    Padding="4" CornerRadius="4">
                <Grid>
                    <StackPanel>
                        <TextBlock Text="Время: 04.05.2021"/>
                        <TextBlock Text="Отправитель: admin"/>
```

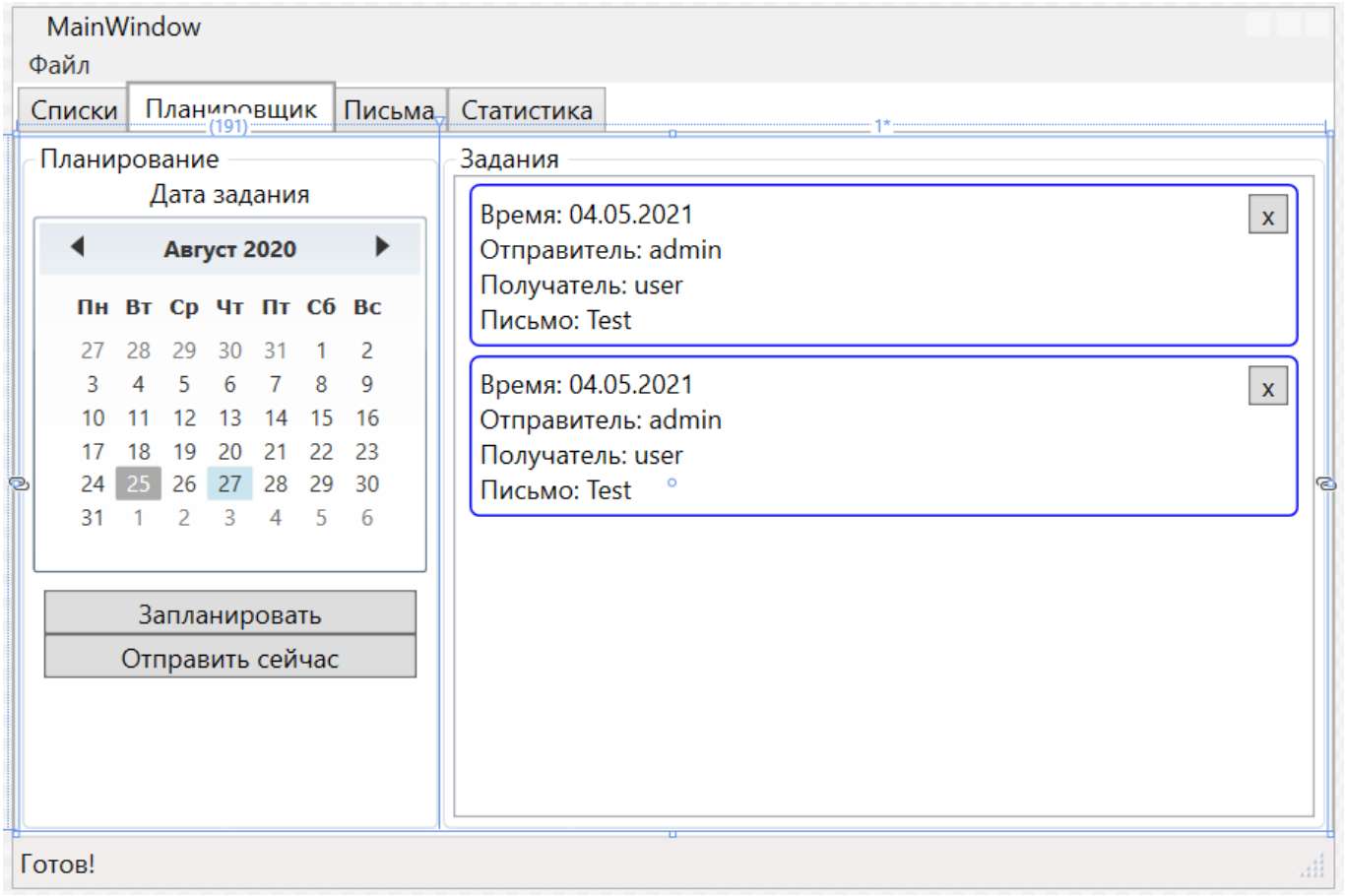
```
                        <TextBlock Text="Получатель: user"/>
                        <TextBlock Text="Письмо: Test"/>
                    </StackPanel>
                    <Button VerticalAlignment="Top"
                            HorizontalAlignment="Right"
                            Padding="5,0"
                            ToolTip="Удалить"
                            Content="x"/>
                </Grid>
            </Border>
        </ListBoxItem>
        <ListBoxItem HorizontalContentAlignment="Stretch">
            <Border BorderBrush="Blue" BorderThickness="1"
                    Padding="4" CornerRadius="4">
                <Grid>
                    <StackPanel>
                        <TextBlock Text="Время: 04.05.2021"/>
                        <TextBlock Text="Отправитель: admin"/>
                        <TextBlock Text="Получатель: user"/>
                        <TextBlock Text="Письмо: Test"/>
                    </StackPanel>
                    <Button VerticalAlignment="Top"
                            HorizontalAlignment="Right"
                            Padding="5,0"
                            ToolTip="Удалить"
                            Content="x"/>
                </Grid>
            </Border>
        </ListBoxItem>
    </ListBox>
</GroupBox>
```

Внутри списка создано два элемента `ListBoxItem` в каждом из которых находится одна и та же разметка: в её основе лежит рамка с указанием цвета (синий), толщина (1 точка), расстояния до содержимого (4 точки с каждой из сторон) и радиуса скругления углов (4 точки). Внутри рамки размещена панель-компоновки `Grid` для того, чтобы можно было добавить одновременно и основное содержимое и поверх этого содержимого вторым слоем вывести кнопку удаления.

Содержимое организовано в виде набора текстовых блоков `TextBlock` сгруппированных вертикально с помощью `StackPanel`.

Для кнопки дополнительно задано свойство `ToolTip="Удалить"`, позволяющее добавить (к любому визуальному элементу) всплывающую подсказку.

Таким образом, на данном этапе вкладка управления планировщиком в дизайнере имеет следующий вид:



На последок добавим визуальный элемент, позволяющий изменять размер колонок макета `Grid` - `GridSplitter`

Разметка получилась сделующая:

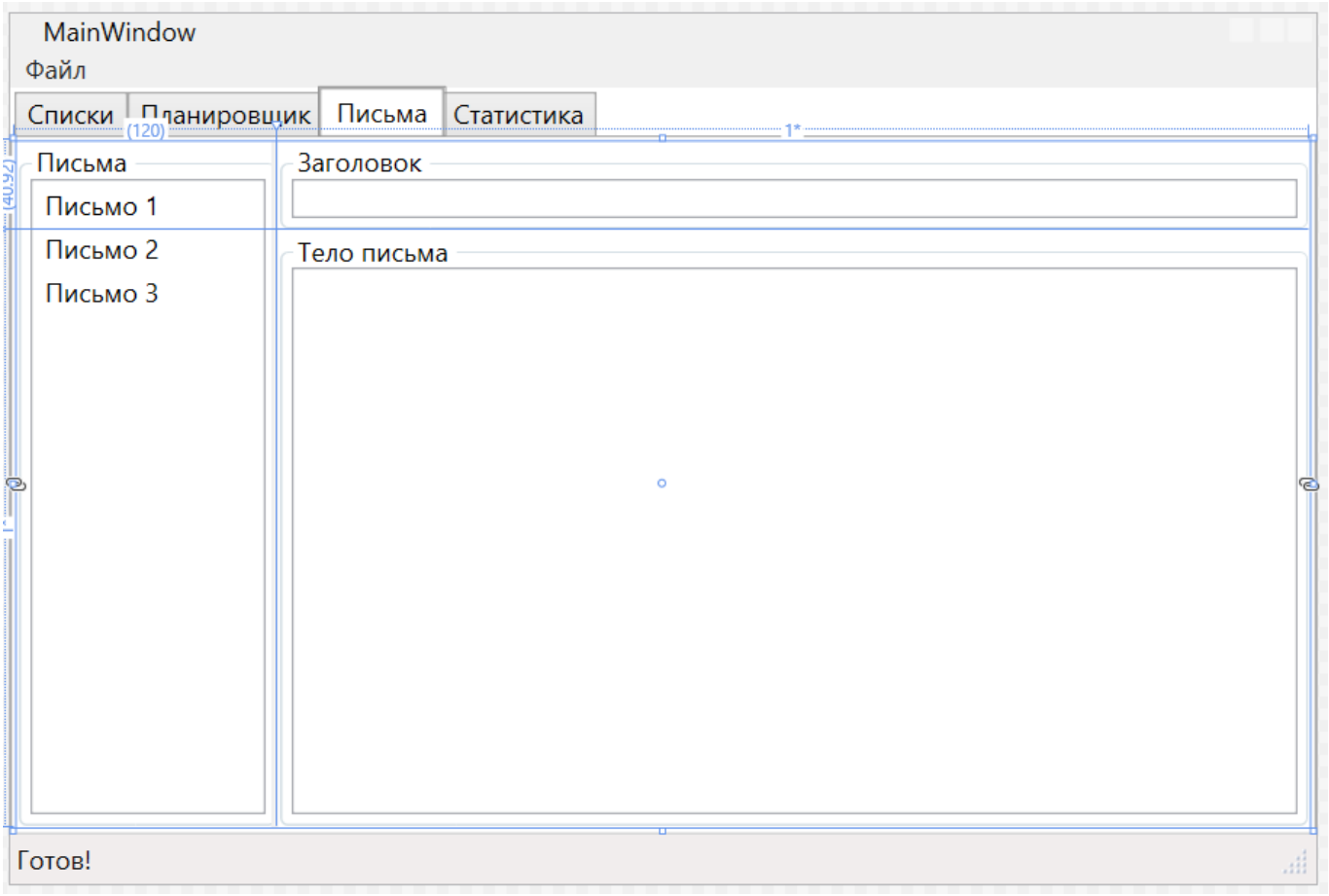
```
<TabItem Header="Планировщик">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>

        <GroupBox Grid.Column="0" Header="Планирование">
            <StackPanel>
                <TextBlock Text="Дата задания"
                           HorizontalAlignment="Center"/>
                <Calendar SelectedDate="2020-08-27"/>
                <UniformGrid Columns="1" Margin="5">
                    <Button Content="Запланировать"/>
                    <Button Content="Отправить сейчас"/>
                </UniformGrid>
            </StackPanel>
        </GroupBox>
        <GroupBox Grid.Column="1" Header="Задания">
            <ListBox>
                <ListBoxItem HorizontalContentAlignment="Stretch">
                    <Border BorderBrush="Blue" BorderThickness="1"
                            Padding="4" CornerRadius="4">
```

```
                        <Grid>
                            <StackPanel>
                                <TextBlock Text="Время: 04.05.2021"/>
                                <TextBlock Text="Отправитель: admin"/>
                                <TextBlock Text="Получатель: user"/>
                                <TextBlock Text="Письмо: Test"/>
                            </StackPanel>
                            <Button VerticalAlignment="Top"
                                    HorizontalAlignment="Right"
                                    Padding="5,0"
                                    ToolTip="Удалить"
                                    Content="x"/>
                        </Grid>
                    </Border>
                </ListBoxItem>
                <ListBoxItem HorizontalContentAlignment="Stretch">
                    <Border BorderBrush="Blue" BorderThickness="1"
                            Padding="4" CornerRadius="4">
                        <Grid>
                            <StackPanel>
                                <TextBlock Text="Время: 04.05.2021"/>
                                <TextBlock Text="Отправитель: admin"/>
                                <TextBlock Text="Получатель: user"/>
                                <TextBlock Text="Письмо: Test"/>
                            </StackPanel>
                            <Button VerticalAlignment="Top"
                                    HorizontalAlignment="Right"
                                    Padding="5,0"
                                    ToolTip="Удалить"
                                    Content="x"/>
                        </Grid>
                    </Border>
                </ListBoxItem>
            </ListBox>
        </GroupBox>
        <GridSplitter Grid.Column="0"
              VerticalAlignment="Stretch"
              HorizontalContentAlignment="Right"
              Width="3"
              Foreground="Transparent"/>
    </Grid>
</TabItem>
```

Теперь создадим разметку редактора писем.

Экран редактора писем

В основе макета данной вкладки будет также использован `Grid` разбитый на две колонки. В левой будет находиться список писем, а в правой - средства для их редактирования (редактор заголовка и редактор тела письма).




```
<TabItem Header="Письма">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto" MinWidth="120"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <GroupBox Grid.Column="0" Grid.Row="0" Grid.RowSpan="2"
                  Header="Письма">
            <ListBox>
                <ListBoxItem>Письмо 1</ListBoxItem>
                <ListBoxItem>Письмо 2</ListBoxItem>
                <ListBoxItem>Письмо 3</ListBoxItem>
            </ListBox>
        </GroupBox>
        <GroupBox Grid.Column="1" Grid.Row="0" Header="Заголовок">
            <TextBox/>
        </GroupBox>
        <GroupBox Grid.Column="1" Grid.Row="1" Header="Тело письма">
            <TextBox AcceptsReturn="True" AcceptsTab="True"/>
        </GroupBox>
    </Grid>
</TabItem>
```

Из особенностей данной разметки следует отметить лишь указание минимальной ширины левой колонки, растягивание некоторых визуальных элементов на две строки макета `Grid` и указание дополнительных свойств для поля ввода текста тела письма, позволяющие использовать клавишу `Enter` для переноса строки текста в нём.

Экран статистики

Добавим во вкладку статистики разметку следующего макета:

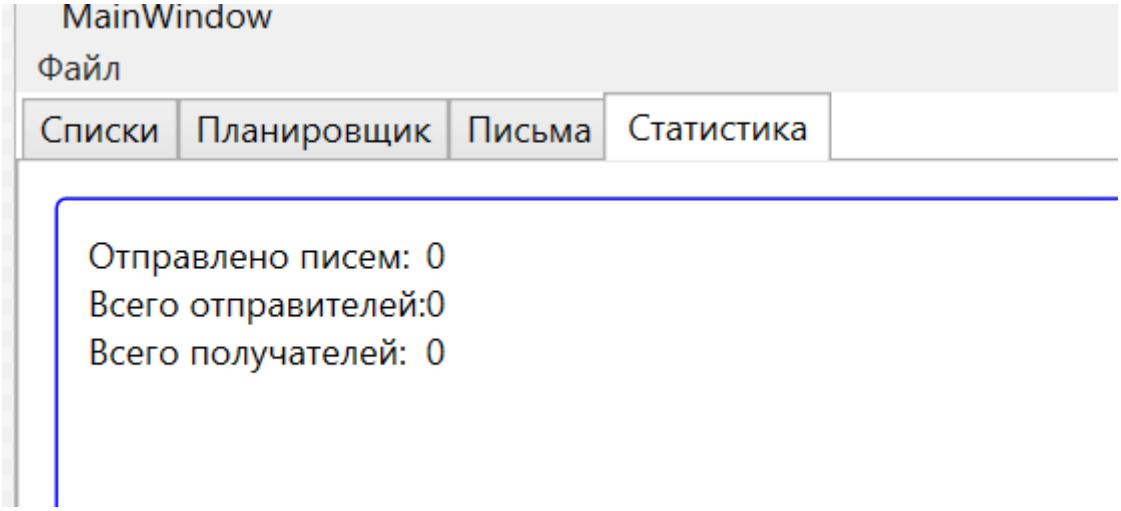
```
<Border Margin="10" Padding="10" BorderThickness="1"
        BorderBrush="Blue" CornerRadius="3">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>

    </Grid>
</Border>
```

Это таблица, содержащая две колонки (ширина левой определяется по размерам содержимого) и три строки (высота каждой строки также автоматически будет подстраиваться под содержимое).

В каждую строку добавим пару элементов `TextBlock`.

```
<Border Margin="10" Padding="10" BorderThickness="1"
        BorderBrush="Blue" CornerRadius="3">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="Auto"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>

        <TextBlock Grid.Column="0" Grid.Row="0" Text="Отправлено писем:"/>
        <TextBlock Grid.Column="1" Grid.Row="0" Text="0"/>

        <TextBlock Grid.Column="0" Grid.Row="1" Text="Всего отправителей:"/>
        <TextBlock Grid.Column="1" Grid.Row="1" Text="0"/>
```

```
        <TextBlock Grid.Column="0" Grid.Row="2" Text="Всего получателей:"/>
        <TextBlock Grid.Column="1" Grid.Row="2" Text="0"/>
    </Grid>
</Border>
```



Данный выведены на экран, но визуально данные выглядят непривлекательно. Можно добавить различные визуальные изменения для каждого из элементов `TextBlock`. Но, во-первых, если элементов будет очень много, то разметка разрастётся до ужасающих размеров, а во-вторых, уже можно увидеть, что хоть все визуальные элементы и являются текстовыми блоками, но их можно разделить на логические группы: левая - имена параметров; правая - значения параметров.

У каждого элемента разметки существует понятие словаря ресурсов `Resources`. В этот словарь можно разместить любые объекты снабжая каждый из них своим уникальным текстовым ключом. При этом, все ресурсы родительского элемента становятся доступными всем его дочерним элементам.

Мы можем в элементе `Grid` разместить в его ресурсах объект `Style`/

```
<Grid>
    <Grid.Resources>
        <Style x:Key="TextBlockStyle"></Style>
    </Grid.Resources>
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition/>
    </Grid.ColumnDefinitions>

    ...
```

Теперь этот стиль, как ресурс, можно применять к отдельным визуальным элементам:

```
<TextBlock Grid.Column="0" Grid.Row="0" Text="Отправлено писем:" Style="
{StaticResource TextBlockStyle}"/>
```

Добавка `Style="{StaticResource TextBlockStyle}"` ко всем элементам управления разметку не украсит. Но можно воспользоваться одной возможностью, которую предоставляет WPF для работы со стилями. Для стилей можно не указывать ключ в словаре ресурсов. Но только в том случае если указан тип для которого формируется стиль. В этом случае платформа сгенерирует ключ автоматически.

Таким образом, мы можем сформировать следующий стиль:

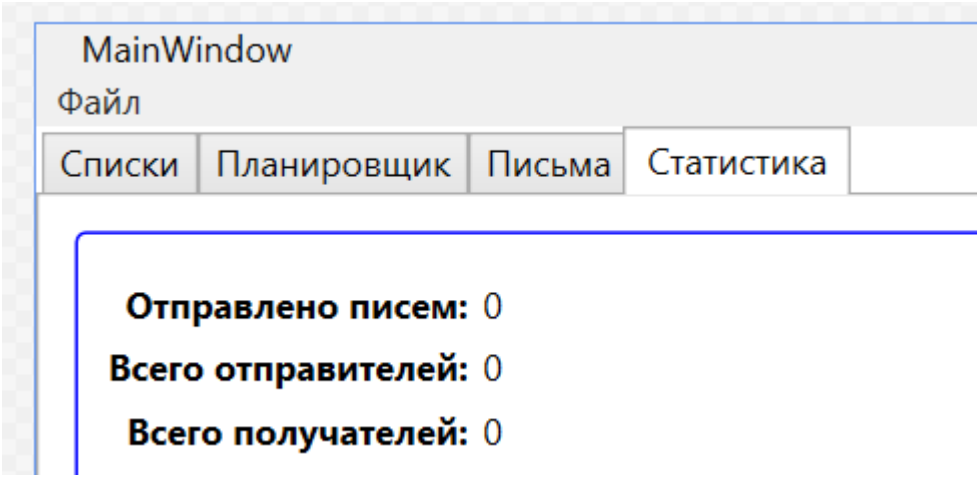
```
<Grid.Resources>
    <Style TargetType="TextBlock">
        <Setter Property="Margin" Value="0,5,0,0"/>
    </Style>
</Grid.Resources>
```

В этом стиле указано, что для текстового блока должно быть установлено свойство "Внешняя рамка" в значение "слева:0; сверху:4; справа:0; снизу:0". Но стиль при этом сейчас применяется у нас ко всем текстовым блокам не различая блоки, расположенные слева от блоков, расположенных справа. Давайте это изменим...

Стили позволяют внутри себя использовать триггеры. У каждого стиля есть коллекция триггеров. Мы можем задать триггер, который будет реагировать на значение нужного нам свойства. Установим особенности для текстовых блоков, у которых значение свойства `Grid.Column="0"`.

```
<Grid.Resources>
    <Style TargetType="TextBlock">
        <Setter Property="Margin" Value="0,5,0,0"/>
        <Style.Triggers>
            <Trigger Property="Grid.Column" Value="0">
                <Setter Property="FontWeight" Value="Bold"/>
                <Setter Property="HorizontalAlignment" Value="Right"/>
                <Setter Property="Margin" Value="0,5,5,0"/>
            </Trigger>
        </Style.Triggers>
    </Style>
</Grid.Resources>
```

В результате мы для всех элементов в данном контейнере `Grid` задали свойство `Margin="0,5,0,0"` и дифференцировано для левой колонки задали своё значение отступов, жирный шрифт и "прижали" элементы к правому краю.



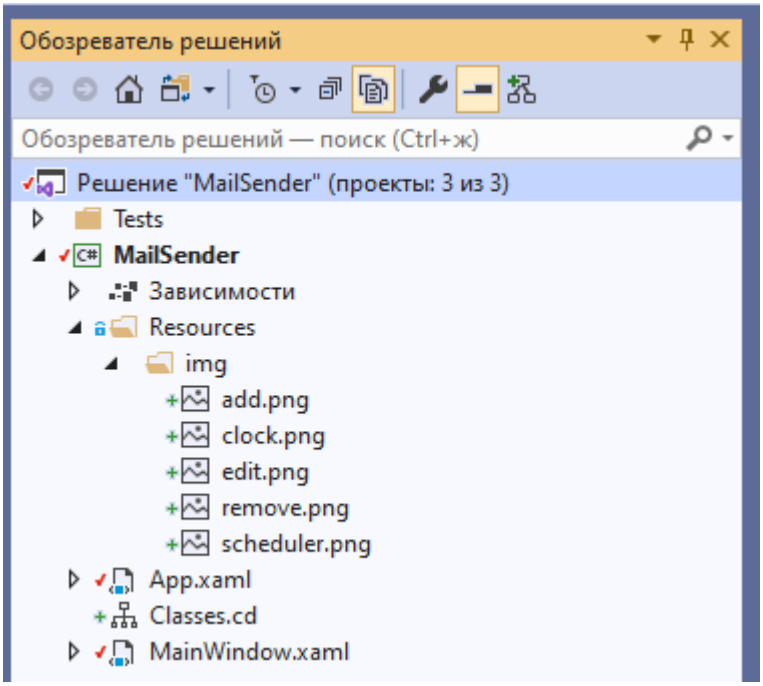
Макет интерфейса на этом этапе можно считать готовым. Приложение стоит скомпилировать и запустить. Надо проверить, во-первых, компилируется ли и запускается оно, и во-вторых, корректно ли ведёт себя макет при измененииии размеров окна.

Использование изображений для улучшения внешнего вида пользовательского интерфейса

Вернёмся на вкладку управления списками.

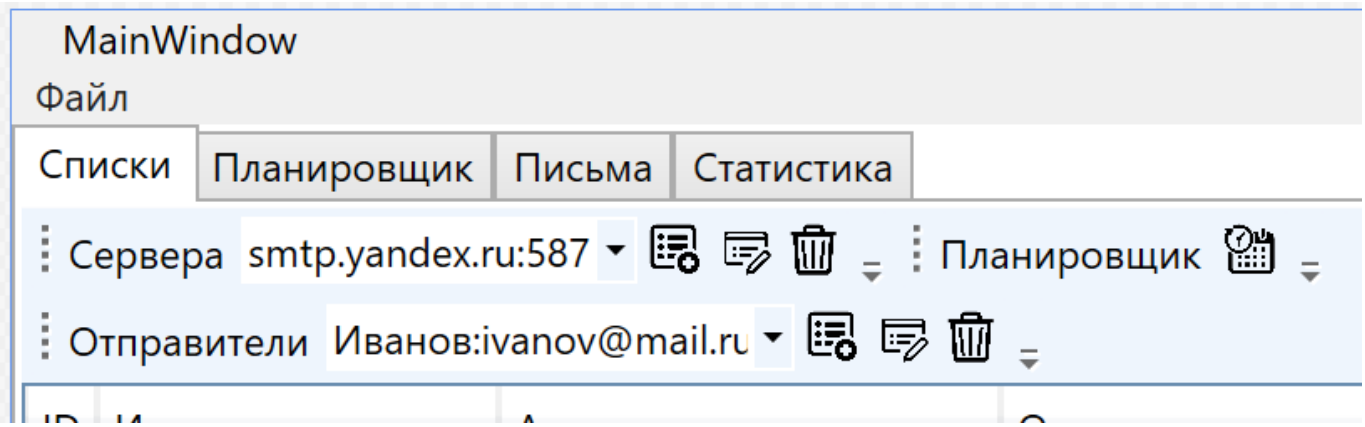
У нас есть панели инструментов для управления списками. На этих панелях есть кнопки с текстом. Для интерфейса пользователя гораздо удобнее и нагляднее если кнопки снабжены картинками - графическими символами операций.

Найдём три картинки олицетворяющие операции добавления, редактирования и удаления элемента в любом графической формате (bmp, jpg, png...) и добавим их в папку `resourdes/img`



Изменим разметку панелей добавив в содержимое кнопок объект `Image`:

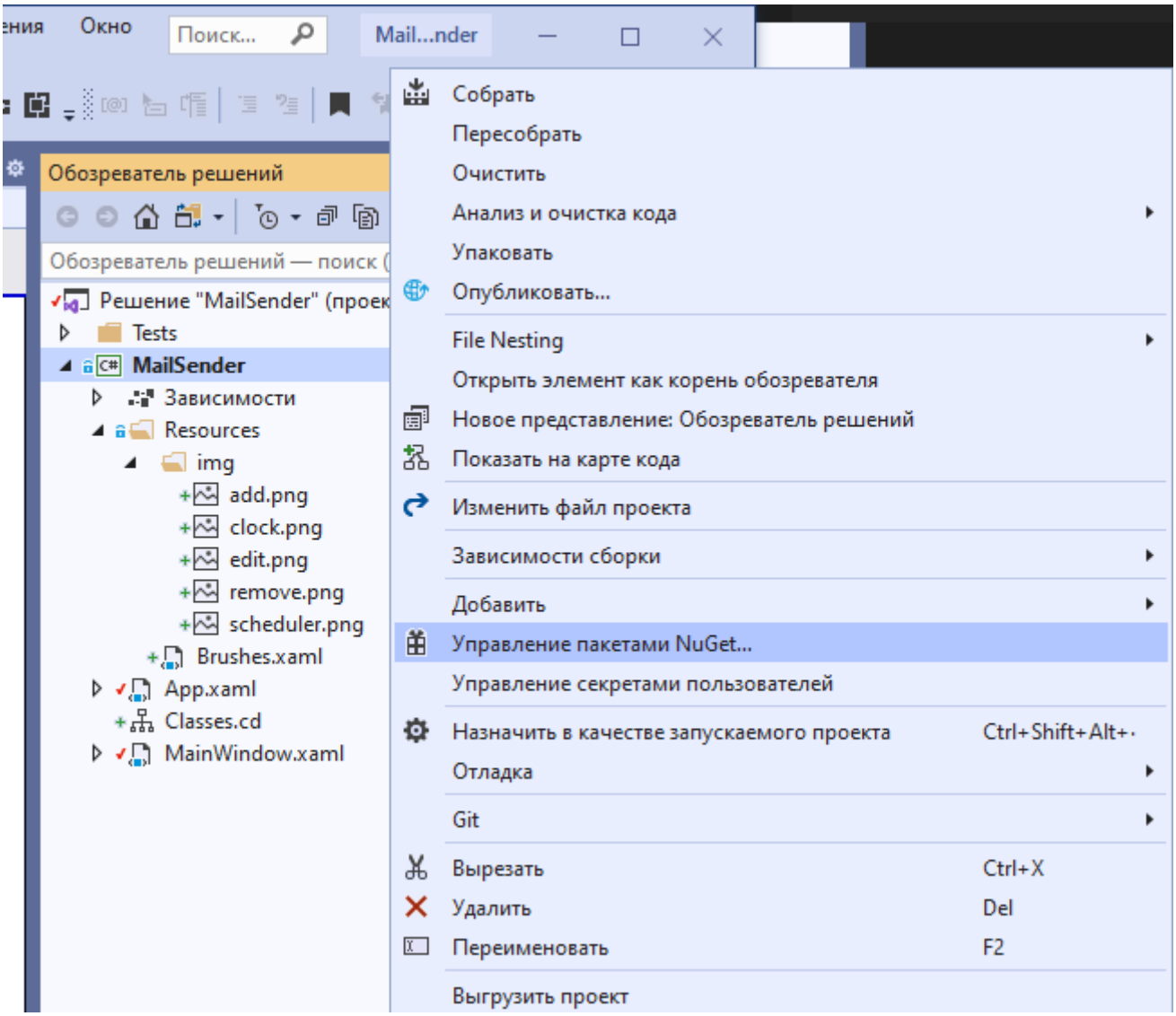
```
<ToolBar Header="Сервера" Name="Servers">
    <ComboBox MinWidth="90" MaxWidth="140" SelectedIndex="0">
        <ComboBoxItem>smtp.yandex.ru:587</ComboBoxItem>
        <ComboBoxItem>smtp.gmail.com</ComboBoxItem>
        <ComboBoxItem>smtp.mail.ru</ComboBoxItem>
    </ComboBox>

    <Button ToolTip="Добавить">
        <Image Source="Resources/img/add.png" Height="15"/>
    </Button>
    <Button ToolTip="Редактировать">
        <Image Source="Resources/img/edit.png" Height="15"/>
    </Button>
    <Button ToolTip="Удалить">
        <Image Source="Resources/img/remove.png" Height="15"/>
    </Button>
</ToolBar>
```



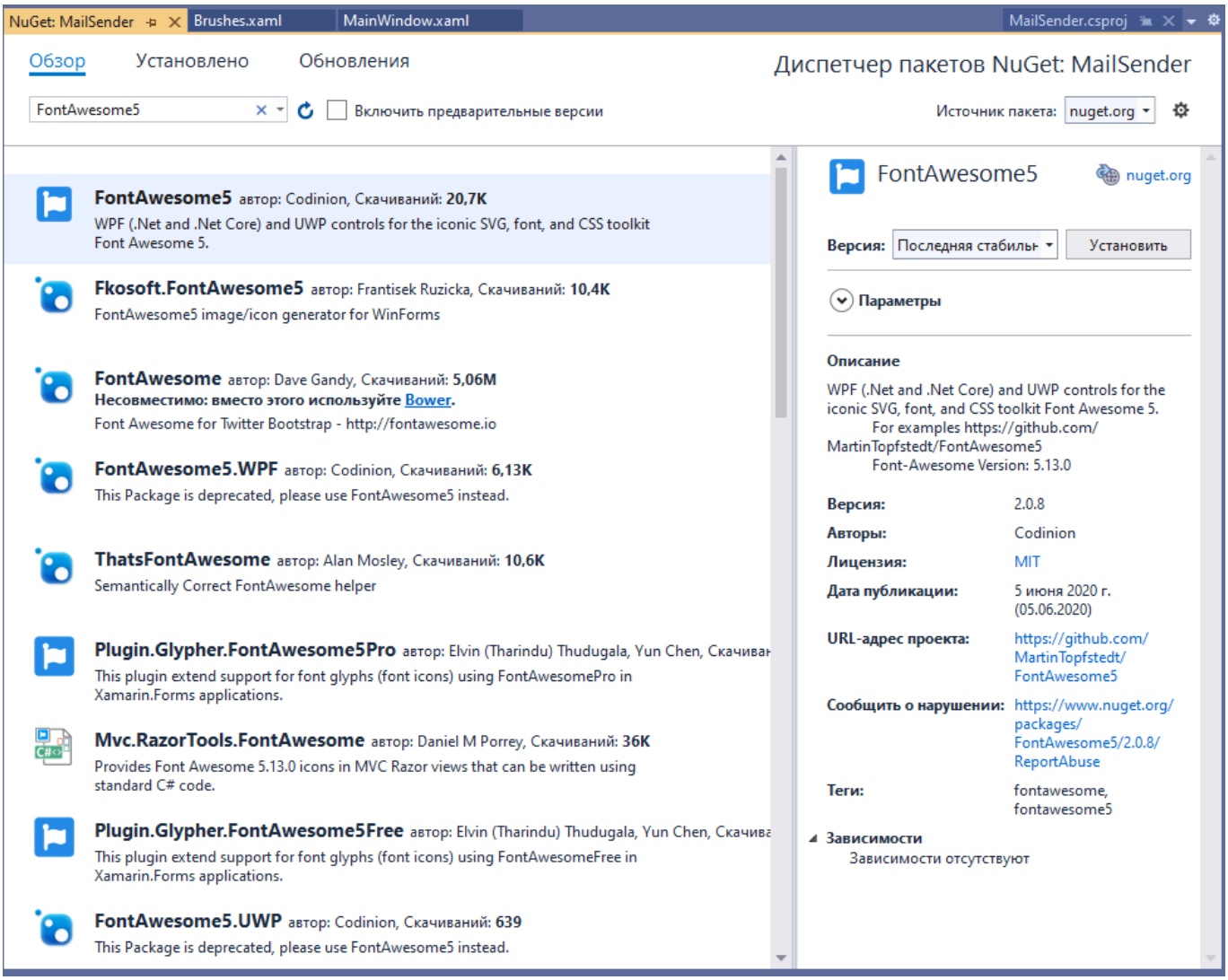
Второй вариант - использование специализированных библиотек, способных генерировать и добавлять изображения значков. Одним из таких библиотек (пакетов) является пакет FontAwesome5. Обычно ссылку на любую необходимую библиотеку можно добавить через диалог управления ссылками. Но для этого придётся вручную найти и скачать файл `.dll`, разместить его в папке проекта/решения и таскать его везде с собой. Также придётся вручную следить за наличием этого файла и обновлять его также придётся вручную при выходе новых версий.

С недавних пор в среде разработки появился специальный модуль - менеджер пакетов NuGet, позволяющий автоматизировать этот процесс.

Откроем визуальный диалог управления пакетами для нашего проекта. Для этого в обозревателе решений найдём наш проект и вызовем у него контекстное меню. В нём нам необходимо выбрать пункт "Управление пакетами NuGet".

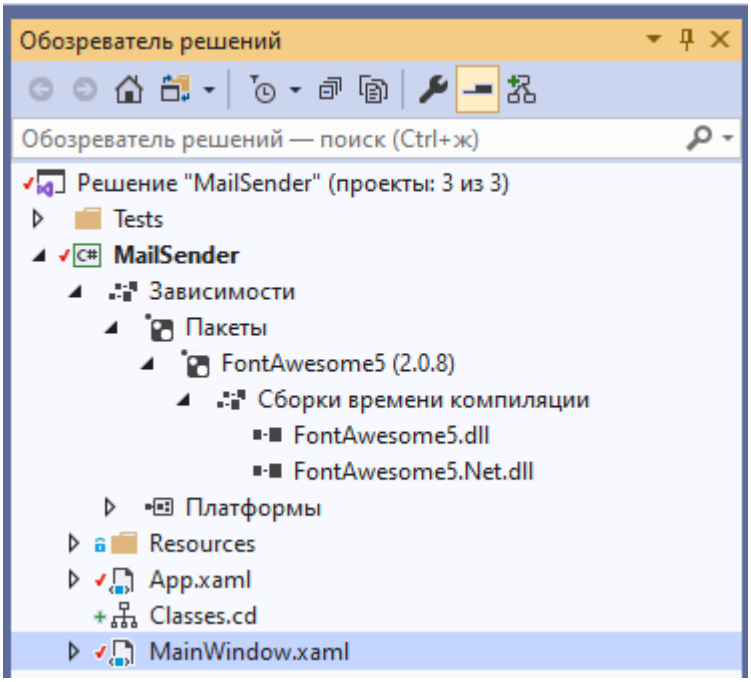


В открывшемся визуальном интерфейсе менеджера пакетов выбираем раздел "Обзор" пакетов и в строку поиска вводим интересующее нас название (доступен поиск пакетов по ключевым словам) "FontAwesome5"



В списке найденных пакетов мы можем найти нужный нам (он здесь первый) и установить его путём нажатия на кнопку "Установить" справа. Также при необходимости можно выбрать нужную версию пакета.

После установки в списке ссылок проекта появляется установленный пакет.

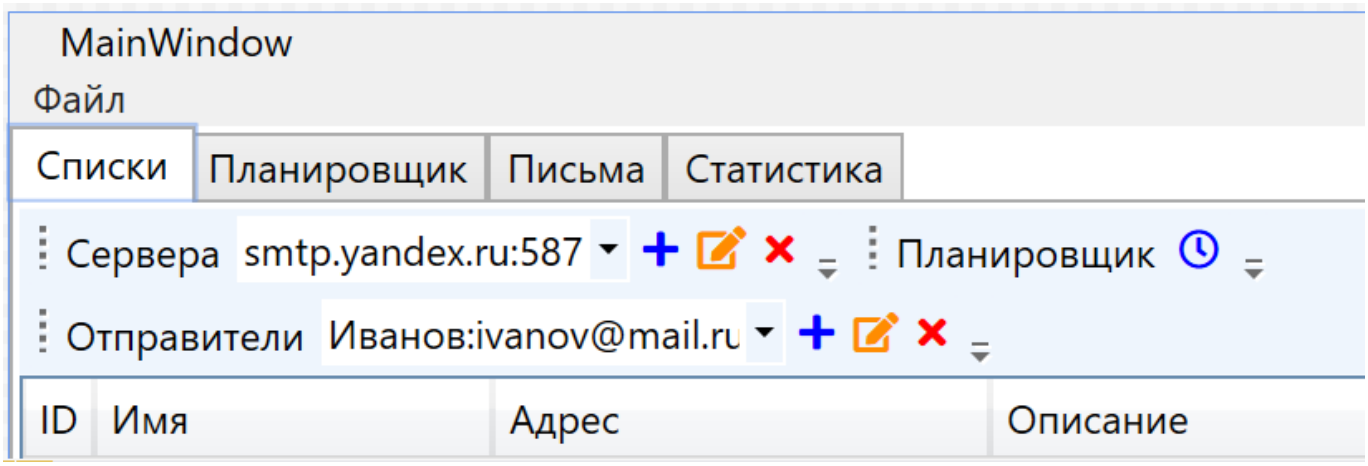


Теперь в разметке мы можем подключить пространство имён этого пакета (Пространство имён для этого пакета было определено его разработчиками. В других пакетах такого может и не быть.)

```
<Window x:Class="MailSender.MainWindow"
        ...
        xmlns:fa="http://schemas.fontawesome.com/icons/"
        ...
        ...>
    <DockPanel>
    ...
    </DockPanel>
</Window>
```

Теперь, определив псевдоним пространства имён `http://schemas.fontawesome.com/icons/`, можно использовать его в разметке.

```
<ToolBar Header="Сервера" Name="Servers">
    <ComboBox MinWidth="90" MaxWidth="140" SelectedIndex="0">
        <ComboBoxItem>smtp.yandex.ru:587</ComboBoxItem>
        <ComboBoxItem>smtp.gmail.com</ComboBoxItem>
        <ComboBoxItem>smtp.mail.ru</ComboBoxItem>
    </ComboBox>

    <Button ToolTip="Добавить"
            fa:Awesome.Content="Solid_Plus"
            Foreground="Blue"/>
    <Button ToolTip="Редактировать">
        <fa:ImageAwesome Icon="Solid_Edit"
                         Height="13"
                         Foreground="DarkOrange"/>
    </Button>
    <Button ToolTip="Удалить"
            fa:Awesome.Content="Solid_Times"
            Foreground="Red"/>
</ToolBar>
```



Итоговая разметка

Таким образом, разметка получилась следующая:

```
<Window x:Class="MailSender.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" mc:Ignorable="d"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:fa="http://schemas.fontawesome.com/icons/"
        xmlns:l="clr-namespace:MailSender"
        Title="MainWindow" Height="400" Width="600">
    <DockPanel>
        <Menu DockPanel.Dock="Top">
            <MenuItem Header="Файл">
                <MenuItem Header="Выход"/>
            </MenuItem>
        </Menu>
        <StatusBar DockPanel.Dock="Bottom">
            <StatusBarItem DockPanel.Dock="Right">
                <ResizeGrip/>
            </StatusBarItem>
            <StatusBarItem>
                <TextBlock Text="Готов!"/>
            </StatusBarItem>
        </StatusBar>
        <TabControl>
            <TabItem Header="Списки">
                <Grid>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition/>
                    </Grid.RowDefinitions>
                    <ToolBarTray Grid.Row="0">
                        <ToolBar Header="Сервера" Name="Servers">
                            <ComboBox MinWidth="90" MaxWidth="140"
                                      SelectedIndex="0">
                                <ComboBoxItem>smtp.yandex.ru:587</ComboBoxItem>
                                <ComboBoxItem>smtp.gmail.com</ComboBoxItem>
                                <ComboBoxItem>smtp.mail.ru</ComboBoxItem>
                            </ComboBox>

                            <Button ToolTip="Добавить"
                                    fa:Awesome.Content="Solid_Plus"
                                    Foreground="Blue"/>
                            <Button ToolTip="Редактировать">
                                <fa:ImageAwesome Icon="Solid_Edit"
                                                 Height="13"
                                                 Foreground="DarkOrange"/>
                            </Button>
                            <Button ToolTip="Удалить"
                                    fa:Awesome.Content="Solid_Times"
```

```
                                Foreground="Red"/>
                    </ToolBar>
                    <ToolBar Header="Отправители" Band="1" BandIndex="0">
                        <ComboBox MinWidth="90" MaxWidth="140"
                                  SelectedIndex="0">
                            <ComboBoxItem>Иванов:ivanov@mail.ru</ComboBoxItem>
                            <ComboBoxItem>Петров:petrov@mail.ru</ComboBoxItem>
                            <ComboBoxItem>
                                Сидоров:sidorov@mail.ru
                            </ComboBoxItem>
                        </ComboBox>

                        <Button ToolTip="Добавить"
                                fa:Awesome.Content="Solid_Plus"
                                Foreground="Blue"/>
                        <Button ToolTip="Редактировать"
                                fa:Awesome.Content="Solid_Edit"
                                Foreground="DarkOrange"/>
                        <Button ToolTip="Удалить"
                                fa:Awesome.Content="Solid_Times"
                                Foreground="Red"/>
                    </ToolBar>
                    <ToolBar Header="Планировщик">
                        <Button ToolTip="Запланировать"
                                fa:Awesome.Content="Regular_Clock"
                                Foreground="Blue"/>
                    </ToolBar>
                </ToolBarTray>
                <DataGrid Grid.Row="1">
                    <DataGrid.Columns>
                        <DataGridTextColumn Header="ID"/>
                        <DataGridTextColumn Header="Имя" MinWidth="120"/>
                        <DataGridTextColumn Header="Адрес" MinWidth="150"/>
                        <DataGridTextColumn Header="Описание" Width="*"/>
                    </DataGrid.Columns>
                </DataGrid>
            </Grid>
        </TabItem>
        <TabItem Header="Планировщик">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto"/>
                    <ColumnDefinition/>
                </Grid.ColumnDefinitions>

                <GroupBox Grid.Column="0" Header="Планирование">
                    <StackPanel>
                        <TextBlock Text="Дата задания"
                                   HorizontalAlignment="Center"/>
                        <Calendar SelectedDate="2020-08-27"/>
                        <UniformGrid Columns="1" Margin="5">
                            <Button Content="Запланировать"/>
                            <Button Content="Отправить сейчас"/>
                        </UniformGrid>
```

```
                    </StackPanel>
                </GroupBox>
                <GroupBox Grid.Column="1" Header="Задания">
                    <ListBox>
                        <ListBoxItem HorizontalContentAlignment="Stretch">
                            <Border BorderBrush="Blue" BorderThickness="1"
                                    Padding="4" CornerRadius="4">
                                <Grid>
                                    <StackPanel>
                                        <TextBlock Text="Время: 04.05.2021"/>
                                        <TextBlock Text="Отправитель: admin"/>
                                        <TextBlock Text="Получатель: user"/>
                                        <TextBlock Text="Письмо: Test"/>
                                    </StackPanel>
                                    <Button VerticalAlignment="Top"
                                            HorizontalAlignment="Right"
                                            Padding="5,0"
                                            ToolTip="Удалить"
                                            Content="x"/>
                                </Grid>
                            </Border>
                        </ListBoxItem>
                        <ListBoxItem HorizontalContentAlignment="Stretch">
                            <Border BorderBrush="Blue" BorderThickness="1"
                                    Padding="4" CornerRadius="4">
                                <Grid>
                                    <StackPanel>
                                        <TextBlock Text="Время: 04.05.2021"/>
                                        <TextBlock Text="Отправитель: admin"/>
                                        <TextBlock Text="Получатель: user"/>
                                        <TextBlock Text="Письмо: Test"/>
                                    </StackPanel>
                                    <Button VerticalAlignment="Top"
                                            HorizontalAlignment="Right"
                                            Padding="5,0"
                                            ToolTip="Удалить"
                                            Content="x"/>
                                </Grid>
                            </Border>
                        </ListBoxItem>
                    </ListBox>
                </GroupBox>
                <GridSplitter Grid.Column="0"
                              VerticalAlignment="Stretch"
                              HorizontalContentAlignment="Right"
                              Width="3"
                              Foreground="Transparent"/>
            </Grid>
        </TabItem>
        <TabItem Header="Письма">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto" MinWidth="120"/>
                    <ColumnDefinition/>
```

```
                </Grid.ColumnDefinitions>
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto"/>
                    <RowDefinition/>
                </Grid.RowDefinitions>
                <GroupBox Grid.Column="0" Grid.Row="0" Grid.RowSpan="2"
                          Header="Письма">
                    <ListBox>
                        <ListBoxItem>Письмо 1</ListBoxItem>
                        <ListBoxItem>Письмо 2</ListBoxItem>
                        <ListBoxItem>Письмо 3</ListBoxItem>
                    </ListBox>
                </GroupBox>
                <GroupBox Grid.Column="1" Grid.Row="0" Header="Заголовок">
                    <TextBox/>
                </GroupBox>
                <GroupBox Grid.Column="1" Grid.Row="1" Header="Тело письма">
                    <TextBox AcceptsReturn="True" AcceptsTab="True"/>
                </GroupBox>
                <GridSplitter Grid.Row="0" Grid.Column="0" Grid.RowSpan="2"
                              VerticalAlignment="Stretch"
                              HorizontalContentAlignment="Right"
                              Width="3"
                              Foreground="Transparent"/>
            </Grid>
        </TabItem>
        <TabItem Header="Статистика">
            <Border Margin="10" Padding="10" BorderThickness="1"
                    BorderBrush="Blue" CornerRadius="3">
                <Grid>
                    <Grid.Resources>
                        <Style TargetType="TextBlock">
                            <Setter Property="Margin" Value="0,5,0,0"/>
                            <Style.Triggers>
                                <Trigger Property="Grid.Column" Value="0">
                                    <Setter Property="FontWeight"
                                            Value="Bold"/>
                                    <Setter Property="HorizontalAlignment"
                                            Value="Right"/>
                                    <Setter Property="Margin"
                                            Value="0,5,5,0"/>
                                </Trigger>
                            </Style.Triggers>
                        </Style>
                    </Grid.Resources>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition Height="Auto"/>
                    </Grid.RowDefinitions>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition/>
                    </Grid.ColumnDefinitions>
```

```
                        <TextBlock Grid.Column="0" Grid.Row="0"
                                   Text="Отправлено писем:"/>
                        <TextBlock Grid.Column="1" Grid.Row="0" Text="0"/>

                        <TextBlock Grid.Column="0" Grid.Row="1"
                                   Text="Всего отправителей:"/>
                        <TextBlock Grid.Column="1" Grid.Row="1" Text="0"/>

                        <TextBlock Grid.Column="0" Grid.Row="2"
                                   Text="Всего получателей:"/>
                        <TextBlock Grid.Column="1" Grid.Row="2" Text="0"/>
                    </Grid>
                </Border>
            </TabItem>
        </TabControl>
    </DockPanel>
</Window>
```

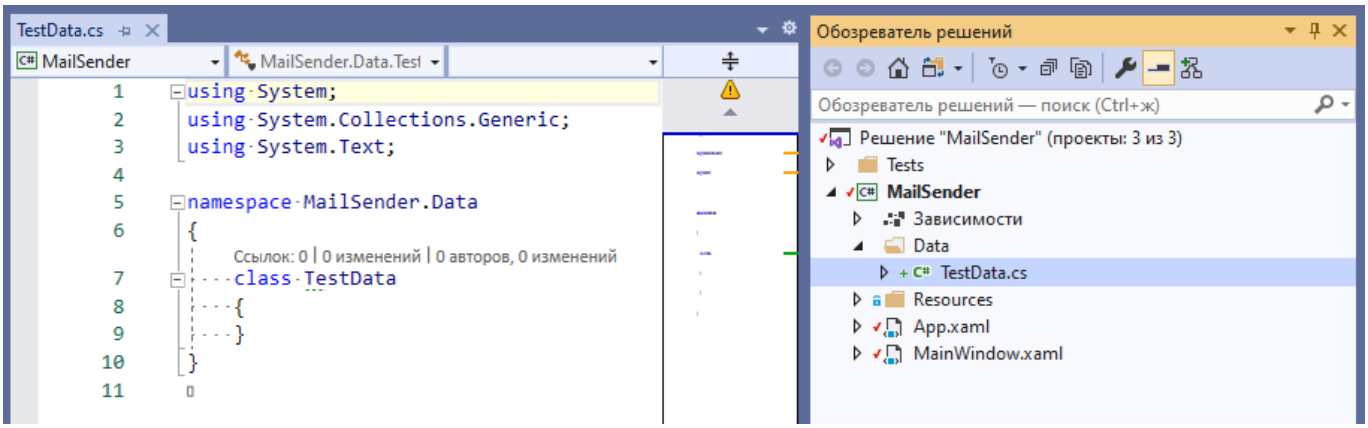
Теперь самое время вдохнуть в эту разметку жизнь.

Бизнес-логика

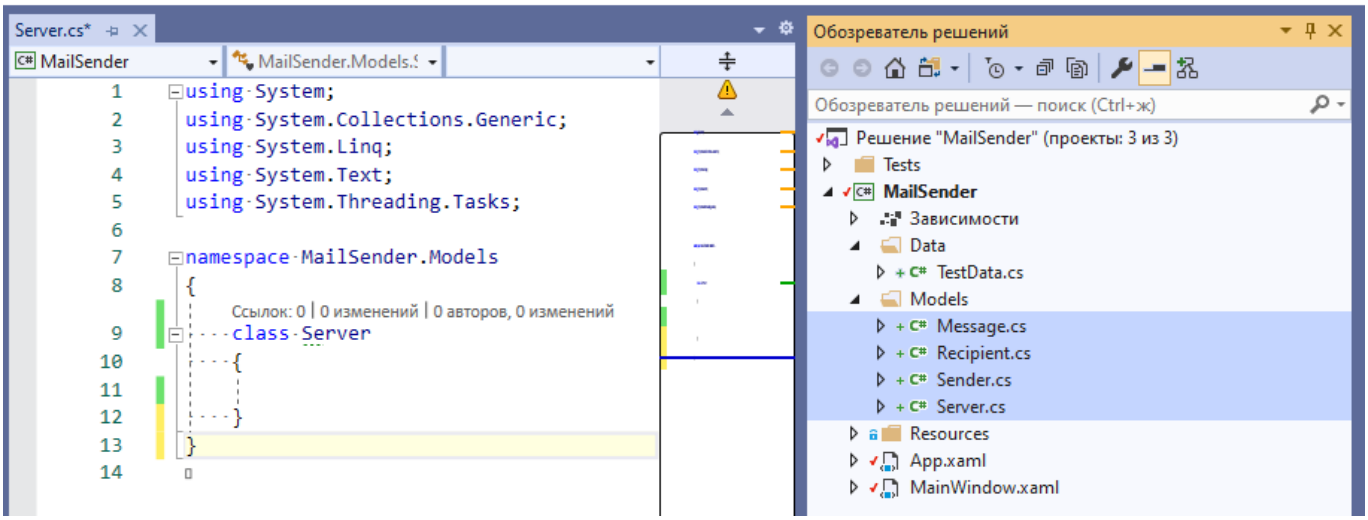
Модели предметной области

Для начала разберёмся с тестовыми данными. Сделаем нормальных источник тестовых данных по серверам, отправителям и получателям.

Для этого в папке проекта заведём подпапку `Data` и в ней создадим статический класс `TestData`.



Также для описания "предметной области" нам потребуются классы-модели данных: "Отправитель", "Получатель", "Сервер", "Сообщение". Для того чтобы наше приложение смогло оперировать этими понятиями нам необходимо их формализовать в виде классов. Для этого заведём ещё одну папку `Models` и внутри неё создадим соответствующие классы.



Дадим определение для каждой из моделей:

Модель почтового сервера

Сервер должен характеризоваться:

0. Идентификатором
1. Именем
2. Адресом
3. Номером порта (по умолчанию должен иметь значение 25)
4. Использовать, Или не использовать SSL (защищённое соединение)
5. Именем пользователя
6. Паролем (желательно не хранить в открытом виде)
7. Комментарием (описанием)

Добавим в класс модели сервера соответствующие свойства

```
namespace MailSender.Models
{
    class Server
    {
        public int Id { get;set; }
        public string Name { get; set; }
        public string Address { get; set; }
        public int Port { get; set; } = 25;
        public bool UseSSL { get; set; }
        public string Login { get; set; }
        public string Password { get; set; }
        public string Description { get; set; }
    }
}
```

Модель отправителя почты

Отправитель почты должен иметь:

0. Идентификатор
1. Имя
2. Адрес
3. Комментарий

```
namespace MailSender.Models
{
    class Sender
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Address { get; set; }
        public string Description { get; set; }
    }
}
```

Модель получателя почты

"Получатель" почты ничем от "отправителя" отличаться не будет. Но вопросы реализации структуры наследования моделей мы пока рассматривать не будет. Просто скопируем структуру свойств (хотя пытливый читатель может и проработать структуру базовых абстрактных классов моделей).

```
namespace MailSender.Models
{
    class Recipient
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Address { get; set; }
        public string Description { get; set; }
    }
}
```

Модель почтового сообщения

Для сообщение важными будут являться лишь два основных понятия: заголовок и тело.

```
namespace MailSender.Models
{
    class Message
    {
        public int Id { get; set; }
        public string Tittle { get; set; }
        public string Body { get; set; }
```

```
    }
}
```

Тестовые данные

После того как система моделей готова, мы можем задать значения тестовых данных.

```
using System.Collections.Generic;
using System.Linq;
using MailSender.Models;

namespace MailSender.Data
{
    static class TestData
    {
        public static IList<Server> Servers { get; } = new List<Server>
        {
            new Server
            {
                Id = 1,
                Name = "Яндекс",
                Address = "smpt.yandex.ru",
                Port = 465,
                UseSSL = true,
                Login = "user@yandex.ru",
                Password = "PassWord",
            },
            new Server
            {
                Id = 2,
                Name = "gMail",
                Address = "smpt.gmail.com",
                Port = 465,
                UseSSL = true,
                Login = "user@yandex.ru",
                Password = "PassWord",
            },
        };

        public static IList<Sender> Senders { get; } = new List<Sender>
        {
            new Sender
            {
                Id = 1,
                Name = "Иванов",
                Address = "ivanov@server.ru",
                Description = "Почта от Иванова"
            },
            new Sender
            {
                Id = 2,
```

```
                Name = "Петров",
                Address = "petrov@server.ru",
                Description = "Почта от Петрова"
            },
            new Sender
            {
                Id = 3,
                Name = "Сидоров",
                Address = "sidorov@server.ru",
                Description = "Почта от Сидорова"
            },
        };

        public static IList<Recipient> Recipients { get; } = new List<Recipient>
        {
            new Recipient
            {
                Id = 1,
                Name = "Иванов",
                Address = "ivanov@server.ru",
                Description = "Почта для Иванова"
            },
            new Recipient
            {
                Id = 2,
                Name = "Петров",
                Address = "petrov@server.ru",
                Description = "Почта для Петрова"
            },
            new Recipient
            {
                Id = 3,
                Name = "Сидоров",
                Address = "sidorov@server.ru",
                Description = "Почта для Сидорова"
            },
        };

        public static IList<Message> Messages { get; } = Enumerable
           .Range(1, 10)
           .Select(i => new Message
            {
                Id = i,
                Tittle = $"Сообщение {i}",
                Body = $"Текст сообщения {i}"
            })
           .ToList();
    }
}
```

Для унификации всех наборов данных используется интерфейс `IList<T>`, позволяющий абстрагироваться от конкретной конструкции набора. Интерфейс представляет собой общее понятие

"списка" - объект, содержимое которого можно перечислять.

Для создания тестовых сообщений здесь используется Linq, позволяющий быстро создать набор данных требуемого размера (размер в 10 сообщений). Принцип его построения следующий:

1. На первом этапе мы формируем перечисление целых чисел от 1 в количестве 10 штук: `Enumerable.Range(1, 10)`
2. На втором этапе мы берём перечисление целых чисел и для каждого из них ставим в соответствие правило, по которому число превращаем в сообщение:

```
i => new Message
{
    Id = i,
    Tittle = $"Сообщение {i}",
    Body = $"Текст сообщения {i}"
}
```

3. Получаемое перечисление сообщений мы заставляем вычислиться и на основе результата требуем сформировать список: `.ToList()`

Теперь, получив класс тестовых данных (нужно выполнить компиляцию проекта чтобы визуальный дизайнер разметки "увидел" этот класс в сформированных бинарниках), мы можем воспользоваться им в разметке. Для этого подключим список тестовых серверов к выпадающему списку серверов в разметке.

Для этого нам понадобится во-первых, добавить псевдонимы пространств имён тестовых данных и моделей в разметку окна:

```
<Window x:Class="MailSender.MainWindow"
        ...
        xmlns:data="clr-namespace:MailSender.Data"
        xmlns:m="clr-namespace:MailSender.Models"
        xmlns:l="clr-namespace:MailSender"
        ...>
    <DockPanel>
        ...
    </DockPanel>
</Window>
```

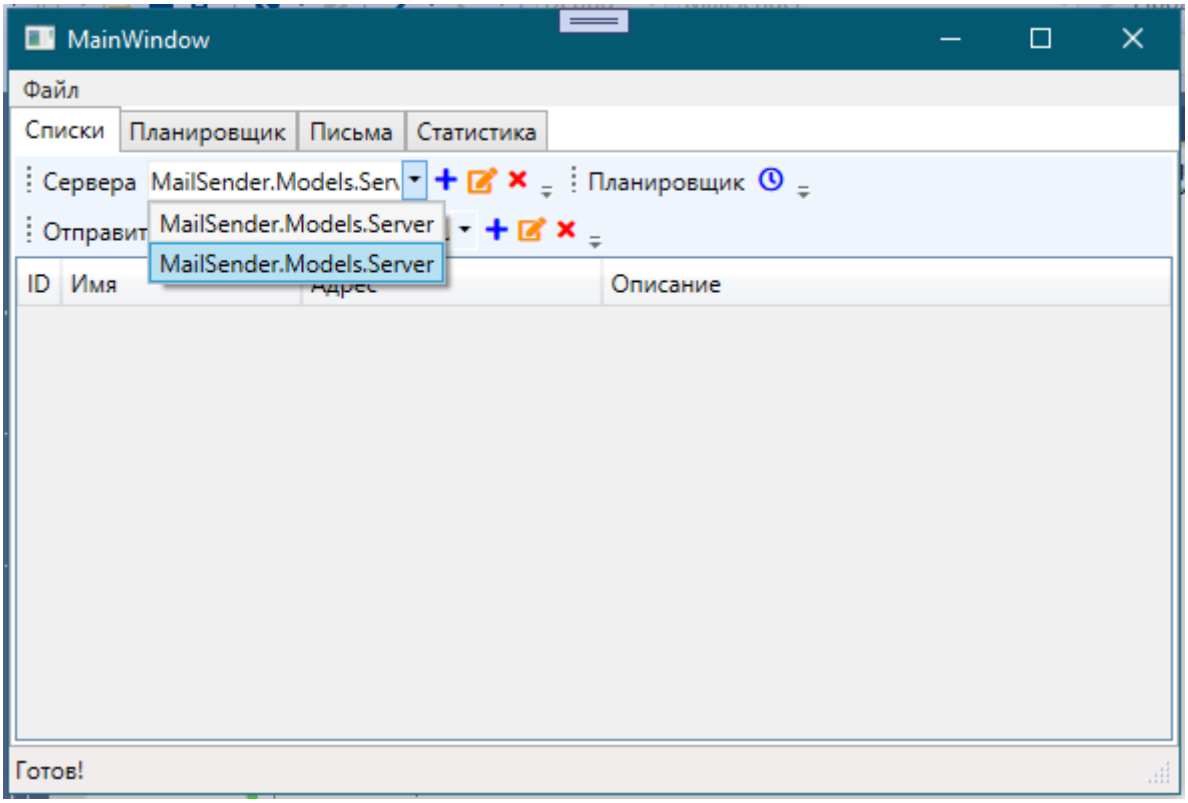
После этого мы можем воспользоваться созданным классом тестовых данных с помощью расширения разметки `{x:Static data:TestData.Servers}`:

```
<ToolBar Header="Сервера" Name="Servers">
    <ComboBox MinWidth="90" MaxWidth="140" SelectedIndex="0"
            ItemsSource="{x:Static data:TestData.Servers}">
    </ComboBox>
```

```
    ...
</ToolBar>
```

Расширение разметки `{x:Static ...}` даёт возможность получить доступ к нужному открытому статическому свойству указанного класса.

В результате в сервера из списка загружаются, но отображаются пока не так как бы этого хотелось:



Исправить это недоразумение можно тремя способами:

1. Переопределить метод `ToString()` у модели "сервера";
2. Использовать свойство `DisplayMemberPath` для выбора имени того свойства, которое `ComboBox` будет отображать на экране;
3. Использовать свойство `ItemTemplate` у `ComboBox` для того чтобы "объяснить" `ComboBox` как нужно визуализировать элемент данных.

Рассмотрим каждый из этих методов:

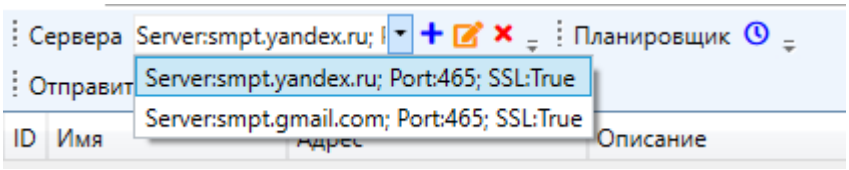
1. Реализация `ToString` у модели

Данный путь является самым "прямым" и (с виду) самым простым для реализации. Дополним определение модели:

```
class Server
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Address { get; set; }
    public int Port { get; set; } = 25;
```

```
        public bool UseSSL { get; set; }
        public string Login { get; set; }
        public string Password { get; set; }
        public string Description { get; set; }

        public override string ToString() =>
            $"Server:{Address}; Port:{Port}; SSL:{UseSSL}";
    }
```

Все визуальные элементы управления, предназначенные для отображения списков, если для них не указан метод визуализации отображаемого элемента данных из списка, они вызывают метод `ToString` и отображают полученный таким образом текст на экране.



Теперь элементы данных списка серверов отображаются луче. Но что делать, если у нас нет доступа к определению класса модели, и мы не можем повлиять на поведение метода `ToString` в нём?

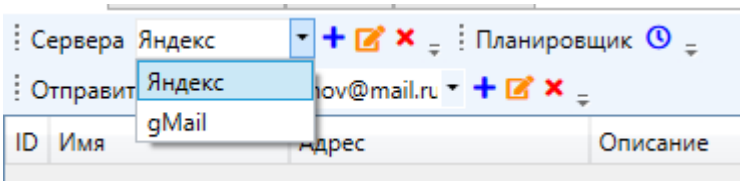
2. Использование `DisplayMemberPath`

Все списочные визуальные элементы управления имеют два свойства, в имени которых есть суффикс `Path`:

- `DisplayMemberPath` - позволяет указать имя свойства для визуализации
- `SelectedValuePath` - позволяет указать имя свойства, значение которого будет вывозиться при выборе элемента в свойстве `SelectedValue`

Укажем нашему `ComboBox`, что при отображении сервером нам надо выводить на экран свойство `Name`.

```
<ComboBox MinWidth="90" MaxWidth="140" SelectedIndex="0"
          ItemsSource="{x:Static data:TestData.Servers}"
          DisplayMemberPath="Name"/>
```



Теперь корректируя исключительно разметку мы получаем возможность управления процессом вывода информации на экран.

Но что делать, если нам требуется сформировать более сложное представление каждого из элементов данных? Для этих целей в платформе WPF существует очень мощный и гибкий механизм формирования визуальной части интерфейса на основе шаблонов.

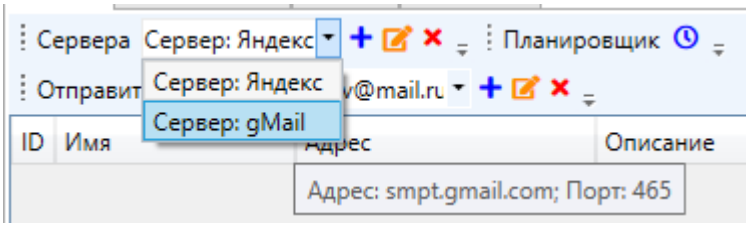
3. Использование шаблонов визуализации данных

Шаблоны в WPF представляют собой блоки разметки, которые применяются визуальным элементов уже в процессе выполнения приложения для того чтобы построить свой внешний вид динамически.

Визуальный интерфейс в WPF строится вокруг двух основных понятий: логического и визуального дерева. Два этих дерева образуются из узлов - объектов, сформированных на основе разметки. Объекты подчиняются друг другу по принципу родительский - дочерние элементы. Логическое дерево определяет логический уровень взаимодействия между элементами интерфейса, а визуальный - "графический". Оба дерева сосуществуют вместе и взаимодействуют между собой. При этом в процессе "жизни" приложения деревья могут меняться. Когда на экране появляется новый элемент (скажем новый элемент в списке), то это приводит к "наращиванию" деревьев. При этом, деревья "наращиваются" сразу целой ветвью, состоящей из ряде уже взаимосвязанных и настроенных узлов (свойства узлов уже выставлены нужным образом). Чтобы определить как именно будет выглядеть добавляемая ветвь мы определяем её шаблон.

Определим шаблон для визуализации одного сервера в списке:

```
<ComboBox MinWidth="90" MaxWidth="140" SelectedIndex="0"
          ItemsSource="{x:Static data:TestData.Servers}">
    <ComboBox.ItemTemplate>
        <DataTemplate>
            <TextBlock>
                <TextBlock.ToolTip>
                    <TextBlock>
                        Адрес:
                        <TextBlock Text="{Binding Address}"/>;
                        Порт:
                        <TextBlock Text="{Binding Port}"/>
                    </TextBlock>
                </TextBlock.ToolTip>
                Сервер: <TextBlock Text="{Binding Name}"/>
            </TextBlock>
        </DataTemplate>
    </ComboBox.ItemTemplate>
</ComboBox>
```



Здесь мы плавно подходим к такому понятию, как "привязка". Здесь устанавливается связь между свойством модели и свойством `TextBlock.Text` элемента разметки.

Сам шаблон данных представляет собой кусочек разметки, который будет вставлен в логическое дерево (а на его основе будет сформировано и визуальное дерево) при визуализации каждого объекта модели "Сервер".

При этом сам шаблон:

```
<DataTemplate>
    <TextBlock>
        <TextBlock.ToolTip>
            <TextBlock>
                Адрес:
                <TextBlock Text="{Binding Address}"/>;
                Порт:
                <TextBlock Text="{Binding Port}"/>
            </TextBlock>
        </TextBlock.ToolTip>
        Сервер: <TextBlock Text="{Binding Name}"/>
    </TextBlock>
</DataTemplate>
```

представляет собой элемент `TextBlock` внутри которого формируется текст `Сервер:` + добавляется текст из вложенного элемента `TextBlock`. Свойство `Text` вложенного элемента связывается со свойством `Name` модели "Сервер". Также для формирования всплывающей подсказки задаётся свойство `ToolTip` как целый блок разметки:

```
<TextBlock>
    Адрес:
    <TextBlock Text="{Binding Address}"/>;
    Порт:
    <TextBlock Text="{Binding Port}"/>
</TextBlock>
```

Разметка всплывающей подсказки также представляет собой текстовый блок, содержимое которого является комбинацией обычного текста и вложенных текстовый блоков. При этом надо учитывать, что каждый перенос строки в такой комбинации будет порождать "пробел" в формируемом тексте.

Использование шаблонов является самым гибким и предпочтительным вариантом формирования визуального представления элементов списков. Из недостатков можно выделить лишь "многословность" получаемой в результате разметки. Но этот недостаток можно нивелировать путём выноса шаблонов в ресурсы.

Сервис рассылки почты

Теперь сформируем класс, объект которого в нашем приложении будет ответственен за главную бизнес-функцию - рассылку почты.

Начнём с простого - создадим класс с параметрами конструктора, достаточными для получения информации для подключения к серверу: адрес, порт, использовать/нет SSL, логин и пароль.

```
namespace MailSender.Services
{
```

```
    class SmtpSender
    {
        private readonly string _Address;
        private readonly int _Port;
        private readonly bool _UseSsl;
        private readonly string _Login;
        private readonly string _Password;

        public SmtpSender(
            string Address, int Port, bool UseSSL,
            string Login, string Password)
        {
            _Address = Address;
            _Port = Port;
            _UseSsl = UseSSL;
            _Login = Login;
            _Password = Password;
        }
    }
}
```

Теперь научим этот сервис рассылать почту. Перенесём логику из нашего экспериментального проекта сюда в метод этого класса:

```
namespace MailSender.Services
{
    class SmtpSender
    {
        private readonly string _Address;
        private readonly int _Port;
        private readonly bool _UseSsl;
        private readonly string _Login;
        private readonly string _Password;

        public SmtpSender(
            string Address, int Port, bool UseSSL,
            string Login, string Password)
        {
            _Address = Address;
            _Port = Port;
            _UseSsl = UseSSL;
            _Login = Login;
            _Password = Password;
        }

        public void Send(
            string From, string To,
            string Title, string Message)
        {
            using var message = new MailMessage(From, To)
            {
```

```
                Subject = Title,
                Body = Message
            };

            using var client = new SmtpClient(_Address, _Port)
            {
                EnableSsl = _UseSsl,
                Credentials = new NetworkCredential(_Login, _Password)
            };

            client.Send(message);
        }
    }
}
```

В созданном классе специально не используются созданные ранее модели для повышения унификации кода. Это класс можно будет использовать с любыми данными в любой форме, так как данные атомизированы до простейших типов (строк, чисел, булевых значений...).

Теперь интегрируем данный класс в приложение заставив работать кнопку отправить.

Интеграция с интерфейсом

Именование элементов интерфейса

Для реализации функции отправки почты нам потребуются данные:

- Сервер
- Отправитель
- Получатель
- Письмо

Тестовые данные для всего этого у нас уже есть. В интерфейсе предусмотрены списочные элементы для отображения и выбора соответствующих значений. В панели серверов даже список значений уже подключён и выведен на экран. Проделаем тоже самое с остальными параметрами.

Список отправителей

Список отправителей организуется аналогично списку серверов. Только элементу управления сразу выдадим имя чтобы с ним можно было общаться из кода.

```
<ComboBox Name="SendersList"
          MinWidth="90" MaxWidth="140" SelectedIndex="0"
          ItemsSource="{x:Static data:TestData.Senders}">
    <ComboBox.ItemTemplate>
        <DataTemplate>
            <TextBlock ToolTip="{Binding Description}">
                <TextBox Text="{Binding Name}"/>:
                <TextBox Text="{Binding Address}"/>
            </TextBlock>
```

```
        </DataTemplate>
    </ComboBox.ItemTemplate>
</ComboBox>
```

Список будет носить имя `SendersList`

Список серверов

Список серверов доработаем лишь добавив ему имя `ServersList`

```
<ComboBox Name="ServersList"
          MinWidth="90" MaxWidth="140" SelectedIndex="0"
          ItemsSource="{x:Static data:TestData.Servers}">
    <ComboBox.ItemTemplate>
        <DataTemplate>
            <TextBlock>
                <TextBlock.ToolTip>
                    <TextBlock>
                        Адрес:
                        <TextBlock Text="{Binding Address}"/>;
                        Порт:
                        <TextBlock Text="{Binding Port}"/>
                    </TextBlock>
                </TextBlock.ToolTip>
                Сервер: <TextBlock Text="{Binding Name}"/>
            </TextBlock>
        </DataTemplate>
    </ComboBox.ItemTemplate>
</ComboBox>
```

####№ Список отправителей

Со списком отправителей всё немного сложнее. Для его отображения мы выбрали элемент `DataGrid`. Это таблица. Элемент управления очень умный и на многое способен сам. Он способен даже самостоятельно определить какие колонки нужны для отображения данных. Но мы ему слегка поможем: определим колонки самостоятельно.

```
<DataGrid Grid.Row="1" Name="RecipientsList"
          ItemsSource="{x:Static data:TestData.Recipients}"
          AutoGenerateColumns="False"
          GridLinesVisibility="Vertical"
          AlternatingRowBackground="PowderBlue"
          VerticalGridLinesBrush="Gray">
    <DataGrid.Columns>
        <DataGridTextColumn Header="ID"
                            Binding="{Binding Id}"/>
        <DataGridTextColumn Header="Имя" MinWidth="120"
                            Binding="{Binding Name}"/>
        <DataGridTextColumn Header="Адрес" MinWidth="150"
```

```
                            Binding="{Binding Address}"/>
        <DataGridTextColumn Header="Описание" Width="*"
                            Binding="{Binding Description}"/>
    </DataGrid.Columns>
</DataGrid>
```

Свойство:

- `AutoGenerateColumns="False"` указывает, что не надо заниматься самодеятельностью, а использовать лишь указанный набор колонок
- `GridLinesVisibility="Vertical"` указываем, что нас интересуют лишь вертикальные разделительные линии
- `AlternatingRowBackground="PowderBlue"` при этом, для разделения мы выбираем альтернативный цвет заливки для чётных строк.
- `VerticalGridLinesBrush="Gray"` ну и чтобы разделительные линии колонок не были столь чёрными, выбираем для них серый цвет

Также задаём набор из 4-х колонок и в виде простейших `DataGridTextColumn` текстовых столбцов и для каждой из них устанавливаем видимый текст в заголовке, задаём ширину и указываем что именно надо отображать в ячейках устанавливая привязки `Binding="{Binding ...}"`. То есть, каждый столбец имеет свойство `Binding`, в значение которого мы устанавливаем привязку, задаваемую как "расширение разметки (на это указывают фигурные скобки) `{Binding ...}`.

##№## Список писем

Остались письма. Для них в разметке окна у нас также имеется свой список. Добавим ему источник данных:

```
<GroupBox Grid.Column="0" Grid.Row="0" Grid.RowSpan="2"
          Header="Письма">
    <ListBox Name="MessagesList"
             ItemsSource="{x:Static data:TestData.Messages}"
             DisplayMemberPath="Tittle"
             SelectedIndex="0"/>
</GroupBox>
```

Задав имя списку писем, мы можем теперь обращаться к нему внутри разметки. Установим связь между выбранным письмом в этом списке и полем редактирования текста заголовка письма (и тела письма за компанию).

```
<GroupBox Grid.Column="1" Grid.Row="0" Header="Заголовок">
    <TextBox Text="{Binding SelectedItem.Tittle, ElementName=MessagesList}"/>
</GroupBox>
<GroupBox Grid.Column="1" Grid.Row="1" Header="Тело письма">
    <TextBox AcceptsReturn="True" AcceptsTab="True"
             Text="{Binding SelectedItem.Body, ElementName=MessagesList}"/>
</GroupBox>
```

Окно редактора серверов

Для обеспечения возможности управления серверами создадим окно пользовательского диалога редактора.

Разметка окна у нас будет следующая:

```
<Window x:Class="MailSender.ServerEditDialog"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" mc:Ignorable="d"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:MailSender"
        Title="Почтовый сервер"
        WindowStartupLocation="CenterOwner"
        Width="330" Height="300">
    <DockPanel Margin="5">
        <DockPanel.Resources>
            <!-- Для всех GroupBox... -->
            <Style TargetType="GroupBox">
                <!-- ... убираем рамку -->
                <Setter Property="BorderThickness" Value="0"/>
            </Style>
            <!-- Для всех Button... -->
            <Style TargetType="Button">
                <!-- Внешняя рамка:     -->
                <!--   лева и справа 7  -->
                <!--   сверху и снизу 0 -->
                <Setter Property="Margin" Value="7,0"/>
                <!-- Внутренняя рамка:   -->
                <!--   лева и справа 30  -->
                <!--   сверху и снизу 10 -->
                <Setter Property="Padding" Value="30,10"/>
            </Style>
        </DockPanel.Resources>

        <!-- Панель с кнопками внизу -->
        <UniformGrid DockPanel.Dock="Bottom"
                     Rows="1"
                     HorizontalAlignment="Right"
                     Margin="0,10,10,10"
                     Button.Click="OnButtonClick">
            <!-- Обработчик события кнопки можно "повесить" на всю панель -->
            <Button Content="Ok" IsDefault="True"/>
            <Button Content="Cancel" IsCancel="True"/>
            <!-- IsDefault - кнопка ассоциируется с клавишей Enter  -->
            <!-- IsCancel  - кнопка ассоциируется с клавишей Escape -->
        </UniformGrid>

        <GroupBox Header="Имя сервера" DockPanel.Dock="Top">
            <TextBox Name="ServerName" x:FieldModifier="private"/>
```

```
        </GroupBox>

        <!-- Рамка здесь используется как контейнер для ориантации в макете -->
        <Border DockPanel.Dock="Top">
            <DockPanel>
                <GroupBox Header="SSL" DockPanel.Dock="Right">
                    <CheckBox Name="ServerSSL"
                              HorizontalAlignment="Center"
                              VerticalAlignment="Center"/>
                </GroupBox>
                <GroupBox Header="Порт" DockPanel.Dock="Right">
                    <!-- В обработчике события запретим ввод "не чисел" -->
                    <TextBox Name="ServerPort" Text="25"
                             PreviewTextInput="OnPortTextInput"/>
                </GroupBox>
                <GroupBox Header="Адрес">
                    <TextBox Name="ServerAddress"/>
                </GroupBox>
            </DockPanel>
        </Border>

        <!-- UniformGrid позволяет содержимое сделать одинакового размера -->
        <UniformGrid DockPanel.Dock="Top" Rows="1">
            <GroupBox Header="Логин">
                <TextBox Name="Login"/>
            </GroupBox>
            <GroupBox Header="Пароль">
                <PasswordBox Name="Password"/>
            </GroupBox>
        </UniformGrid>

        <GroupBox Header="Описание">
            <TextBox Name="ServerDescription"/>
        </GroupBox>
    </DockPanel>
</Window>
```

Класс окна будет иметь следующий вид:

```
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;

namespace MailSender
{
    public partial class ServerEditDialog
    {
        /// <summary>
        /// Конструктор окна - формирует внешний вид с помощью
        /// <see cref="InitializeComponent"/>
        /// Конструктор делам приватным чтобы окно можно было создать только одним
```

```
        /// из методов, объявленных внутри класса этого окна. Инкапсуляция...
        /// </summary>
        private  ServerEditDialog() => InitializeComponent();

        /// <summary>
        /// Обработчик события ввода текста
        /// Блокирует ввод нечисловых данных
        /// </summary>
        private void OnPortTextInput(object Sender, TextCompositionEventArgs E)
        {
            // Если источник события - не текстовое поле ввода
            // или текст в поле ввода отсутствует, то...
            // ничего не делаем
            if (!(Sender is TextBox text_box) || text_box.Text == "") return;
            // иначе если не удалось превратить текст в число, то
            // отмечаем событие как обработанное - текст не введётся
            E.Handled = !int.TryParse(text_box.Text, out _);
        }

        /// <summary>
        /// Обработчик события кнопки
        /// Если кнопка IsCancel == true, то результатом диалога будет false
        /// </summary>
        private void OnButtonClick(object Sender, RoutedEventArgs E)
        {
            DialogResult = !((Button)E.OriginalSource).IsCancel;
            Close();
        }

        // Добавляем статические методы для удобства работы с диалогом

        /// <summary>
        /// Метод, позволяющий отобразить диалог для редактирования данных
        /// Редактируемые параметры передаются по ссылке
        /// Если пользователь выбрал Ok, то метод возвращает true
        /// Если пользователь выбрал Cancel, то параметры не меняются.
        /// </summary>
        public static bool ShowDialog(
            string Title, ref string Name,
            ref string Address, ref int Port, ref bool UseSSL,
            ref string Description,
            ref string Login, ref string Password)
        {
            // Создаём окно и инициализируем его свойства
            var window = new ServerEditDialog
            {
                Title = Title,
                // Так можно инициализировать свойства вложенных объектов
                ServerName = { Text = Name },
                ServerAddress = { Text = Address },
                ServerPort = { Text = Port.ToString() },
                ServerSSL = { IsChecked = UseSSL },
                Login = { Text = Login },
                Password = { Password = Password },
```

```
                ServerDescription = { Text = Description },
                // Берём класс "Приложение"
                Owner = Application
                    // получаем экземпляр текущего приложения
                    .Current
                    // берём все окна приложения
                    .Windows
                    // переводим их из интерфейса IEnumerable в IEnumerable<Window>
                    .Cast<Window>()
                    // находим первое окно, у которого свойство IsActive == true
                    .FirstOrDefault(window => window.IsActive)
            };

            if (window.ShowDialog() != true) return false;

            Name = window.ServerName.Text;
            Address = window.ServerAddress.Text;
            Port = int.Parse(window.ServerPort.Text);
            Login = window.Login.Text;
            Password = window.Password.Password;

            return true;
        }

        /// <summary>
        /// Метод, позволяющий отобразить диалог создания нового сервера
        /// Редактируемые параметры передаются по ссылке
        /// Если пользователь выбрал Ok, то метод возвращает true
        /// Если пользователь выбрал Cancel, то возвращаются дефолтные значения
        /// </summary>
        public static bool Create(
            out string Name,
            out string Address,
            out int Port,
            out bool UseSSL,
            out string Description,
            out string Login,
            out string Password)
        {
            // Инициализируем переменные значениями на случай отмены операции
            Name = null;
            Address = null;
            Port = 25;
            UseSSL = false;
            Description = null;
            Login = null;
            Password = null;

            return ShowDialog("Создать сервер",
                ref Name,
                ref Address,
                ref Port,
                ref UseSSL,
                ref Description,
```

```
                ref Login,
                ref Password);
        }
    }
}
```

Ну и осталось последнее - добавить возможность отправки почты. Для этого добавим обработчик событий в кнопке отправки почты: `<Button Content="Отправить сейчас" Click="OnSendNowButtonClick"/>`

```
/// <summary>Обработчик события кнопки немедленной отправки поты</summary>
private void OnSendNowButtonClick(object Sender, RoutedEventArgs E)
{
    // Извлекаем исходные параметры по возможности
    if (!(SendersList.SelectedItem    is Sender sender))       return;
    if (!(RecipientsList.SelectedItem is Recipient recipient)) return;
    if (!(ServersList.SelectedItem    is Server server))       return;
    if (!(MessagesList.SelectedItem   is Message message))     return;
    // Если одни из параметров невозможно получить, то выходим

    // Создаём объект-рассыльщик и заполняем параметры сервера
    var mail_sender = new SmtpSender(
        server.Address, server.Port, server.UseSSL,
        server.Login, server.Password);

    // При отправке почты может возникнуть проблема. Ставим перехват исключения.
    try
    {
        // Запускаем таймер
        var timer = Stopwatch.StartNew();
        // И запускаем процесс отправки почты
        mail_sender.Send(
            sender.Address, recipient.Address,
            message.Tittle, message.Body);
        timer.Stop(); // По завершении останавливаем таймер

        // Если почта успешно отправлена, то отображаем диалоговое окно
        MessageBox.Show(
            $"Почта успешно отправлена за {timer.Elapsed.TotalSeconds:0.##}c",
            "Отправка почты",
            MessageBoxButton.OK,
            MessageBoxImage.Information);
    }
    // Если случилась ошибка, то перехватываем исключение
    catch (SmtpException) // Перехватывает строго нужное нам исключение!
    {
        MessageBox.Show(
            "Ошибка при отправке почты",
            "Отправка почты",
            MessageBoxButton.OK,
            MessageBoxImage.Error);
```

```
    }
}
```

Итоговое окно

Разметка Главного окна

```
<Window x:Class="MailSender.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" mc:Ignorable="d"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:fa="http://schemas.fontawesome.com/icons/"
        xmlns:data="clr-namespace:MailSender.Data"
        xmlns:m="clr-namespace:MailSender.Models"
        xmlns:l="clr-namespace:MailSender"
        Title="Рассыльщик" Height="400" Width="600">
    <DockPanel>
        <Menu DockPanel.Dock="Top">
            <MenuItem Header="Файл">
                <MenuItem Header="Выход"/>
            </MenuItem>
        </Menu>
        <StatusBar DockPanel.Dock="Bottom">
            <StatusBarItem DockPanel.Dock="Right">
                <ResizeGrip/>
            </StatusBarItem>
            <StatusBarItem>
                <TextBlock Text="Готов!"/>
            </StatusBarItem>
        </StatusBar>
        <TabControl>
            <TabItem Header="Списки">
                <Grid>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition/>
                    </Grid.RowDefinitions>
                    <ToolBarTray Grid.Row="0">
                        <ToolBar Header="Сервера">
                            <ComboBox Name="ServersList"
                                   MinWidth="90" MaxWidth="180"
                                   SelectedIndex="0"
                                   ItemsSource="{x:Static data:TestData.Servers}">
                                <ComboBox.ItemTemplate>
                                    <DataTemplate>
                                        <TextBlock>
                                            <TextBlock.ToolTip>
                                                <TextBlock>
                                                    Адрес:
                                                    <TextBlock
```

```
                                        Text="{Binding Address}"/>;
                                    Порт:
                                    <TextBlock
                                        Text="{Binding Port}"/>
                                </TextBlock>
                            </TextBlock.ToolTip>
                            <TextBlock Text="{Binding Name}"/>
                            (<TextBlock Text="{Binding Address}"
                                        Opacity="0.6"/>:
                            <TextBlock Text="{Binding Port}"
                                       Opacity="0.6"/>)
                        </TextBlock>
                    </DataTemplate>
                </ComboBox.ItemTemplate>
            </ComboBox>

            <Button ToolTip="Добавить"
                    Click="OnAddServerButtonClick"
                    fa:Awesome.Content="Solid_Plus"
                    Foreground="Blue"/>
            <Button ToolTip="Редактировать"
                    Click="OnEditServerButtonClick">
                <fa:ImageAwesome Icon="Solid_Edit"
                                 Height="13"
                                 Foreground="DarkOrange"/>
            </Button>
            <Button ToolTip="Удалить"
                    Click="OnDeleteServerButtonClick"
                    fa:Awesome.Content="Solid_Times"
                    Foreground="Red"/>
        </ToolBar>
        <ToolBar Header="Отправители" Band="1" BandIndex="0">
            <ComboBox Name="SendersList"
                   MinWidth="90" MaxWidth="160"
                   SelectedIndex="0"
                   ItemsSource="{x:Static data:TestData.Senders}">
                <ComboBox.ItemTemplate>
                    <DataTemplate>
                       <TextBlock ToolTip="{Binding Description}">
                           <TextBlock Text="{Binding Name}"/>:
                           <TextBlock Text="{Binding Address}"/>
                       </TextBlock>
                    </DataTemplate>
                </ComboBox.ItemTemplate>
            </ComboBox>

            <Button ToolTip="Добавить"
                    fa:Awesome.Content="Solid_Plus"
                    Foreground="Blue"/>
            <Button ToolTip="Редактировать"
                    fa:Awesome.Content="Solid_Edit"
                    Foreground="DarkOrange"/>
            <Button ToolTip="Удалить"
                    fa:Awesome.Content="Solid_Times"
```

```
                                Foreground="Red"/>
                    </ToolBar>
                    <ToolBar Header="Планировщик">
                        <Button ToolTip="Запланировать"
                                fa:Awesome.Content="Regular_Clock"
                                Foreground="Blue"/>
                    </ToolBar>
                </ToolBarTray>
                <DataGrid Grid.Row="1" Name="RecipientsList"
                          ItemsSource="{x:Static data:TestData.Recipients}"
                          AutoGenerateColumns="False"
                          GridLinesVisibility="Vertical"
                          AlternatingRowBackground="PowderBlue"
                          VerticalGridLinesBrush="Gray">
                    <DataGrid.Columns>
                        <DataGridTextColumn Header="ID"
                                            Binding="{Binding Id}"/>
                        <DataGridTextColumn Header="Имя" MinWidth="120"
                                            Binding="{Binding Name}"/>
                        <DataGridTextColumn Header="Адрес" MinWidth="150"
                                            Binding="{Binding Address}"/>
                        <DataGridTextColumn Header="Описание" Width="*"
                                            Binding="{Binding Description}"/>
                    </DataGrid.Columns>
                </DataGrid>
            </Grid>
        </TabItem>
        <TabItem Header="Планировщик">
            <Grid>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="Auto"/>
                    <ColumnDefinition/>
                </Grid.ColumnDefinitions>

                <GroupBox Grid.Column="0" Header="Планирование">
                    <StackPanel>
                        <TextBlock Text="Дата задания"
                                   HorizontalAlignment="Center"/>
                        <Calendar SelectedDate="2020-08-27"/>
                        <UniformGrid Columns="1" Margin="5">
                            <Button Content="Запланировать"/>
                            <Button Content="Отправить сейчас"
                                    Click="OnSendNowButtonClick"/>
                        </UniformGrid>
                    </StackPanel>
                </GroupBox>
                <GroupBox Grid.Column="1" Header="Задания">
                    <ListBox>
                        <ListBoxItem HorizontalContentAlignment="Stretch">
                            <Border BorderBrush="Blue" BorderThickness="1"
                                    Padding="4" CornerRadius="4">
                                <Grid>
                                    <StackPanel>
                                        <TextBlock Text="Время: 04.05.2021"/>
```

```
                                    <TextBlock Text="Отправитель: admin"/>
                                    <TextBlock Text="Получатель: user"/>
                                    <TextBlock Text="Письмо: Test"/>
                                </StackPanel>
                                <Button VerticalAlignment="Top"
                                        HorizontalAlignment="Right"
                                        Padding="5,0"
                                        ToolTip="Удалить"
                                        Content="x"/>
                            </Grid>
                        </Border>
                    </ListBoxItem>
                    <ListBoxItem HorizontalContentAlignment="Stretch">
                        <Border BorderBrush="Blue" BorderThickness="1"
                                Padding="4" CornerRadius="4">
                            <Grid>
                                <StackPanel>
                                    <TextBlock Text="Время: 04.05.2021"/>
                                    <TextBlock Text="Отправитель: admin"/>
                                    <TextBlock Text="Получатель: user"/>
                                    <TextBlock Text="Письмо: Test"/>
                                </StackPanel>
                                <Button VerticalAlignment="Top"
                                        HorizontalAlignment="Right"
                                        Padding="5,0"
                                        ToolTip="Удалить"
                                        Content="x"/>
                            </Grid>
                        </Border>
                    </ListBoxItem>
                </ListBox>
            </GroupBox>
            <GridSplitter Grid.Column="0"
                          VerticalAlignment="Stretch"
                          HorizontalContentAlignment="Right"
                          Width="3"
                          Foreground="Transparent"/>
        </Grid>
    </TabItem>
    <TabItem Header="Письма">
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" MinWidth="120"/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto"/>
                <RowDefinition/>
            </Grid.RowDefinitions>
            <GroupBox Grid.Column="0" Grid.Row="0" Grid.RowSpan="2"
                      Header="Письма">
                <ListBox Name="MessagesList"
                         ItemsSource="{x:Static data:TestData.Messages}"
                         DisplayMemberPath="Tittle"
```

```
                            SelectedIndex="0"/>
                </GroupBox>
                <GroupBox Grid.Column="1" Grid.Row="0" Header="Заголовок">
                    <TextBox Text="{Binding SelectedItem.Tittle,
                                    ElementName=MessagesList}"/>
                </GroupBox>
                <GroupBox Grid.Column="1" Grid.Row="1" Header="Тело письма">
                    <DockPanel>
                        <TextBox AcceptsReturn="True" AcceptsTab="True"
                                 Text="{Binding SelectedItem.Body,
                                        ElementName=MessagesList}"/>
                    </DockPanel>
                </GroupBox>
                <GridSplitter Grid.Row="0" Grid.Column="0" Grid.RowSpan="2"
                              VerticalAlignment="Stretch"
                              HorizontalContentAlignment="Right"
                              Width="3"
                              Foreground="Transparent"/>
            </Grid>
        </TabItem>
        <TabItem Header="Статистика">
            <Border Margin="10" Padding="10" BorderThickness="1"
                    BorderBrush="Blue" CornerRadius="3">
                <Grid>
                    <Grid.Resources>
                        <Style TargetType="TextBlock">
                            <Setter Property="Margin" Value="0,5,0,0"/>
                            <Style.Triggers>
                                <Trigger Property="Grid.Column" Value="0">
                                    <Setter Property="FontWeight"
                                            Value="Bold"/>
                                    <Setter Property="HorizontalAlignment"
                                            Value="Right"/>
                                    <Setter Property="Margin"
                                            Value="0,5,5,0"/>
                                </Trigger>
                            </Style.Triggers>
                        </Style>
                    </Grid.Resources>
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition Height="Auto"/>
                    </Grid.RowDefinitions>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition/>
                    </Grid.ColumnDefinitions>

                    <TextBlock Grid.Column="0" Grid.Row="0"
                               Text="Отправлено писем:"/>
                    <TextBlock Grid.Column="1" Grid.Row="0" Text="0"/>

                    <TextBlock Grid.Column="0" Grid.Row="1"
```

```
                                    Text="Всего отправителей:"/>
                        <TextBlock Grid.Column="1" Grid.Row="1" Text="0"/>

                        <TextBlock Grid.Column="0" Grid.Row="2"
                                   Text="Всего получателей:"/>
                        <TextBlock Grid.Column="1" Grid.Row="2" Text="0"/>
                    </Grid>
                </Border>
            </TabItem>
        </TabControl>
    </DockPanel>
</Window>
```

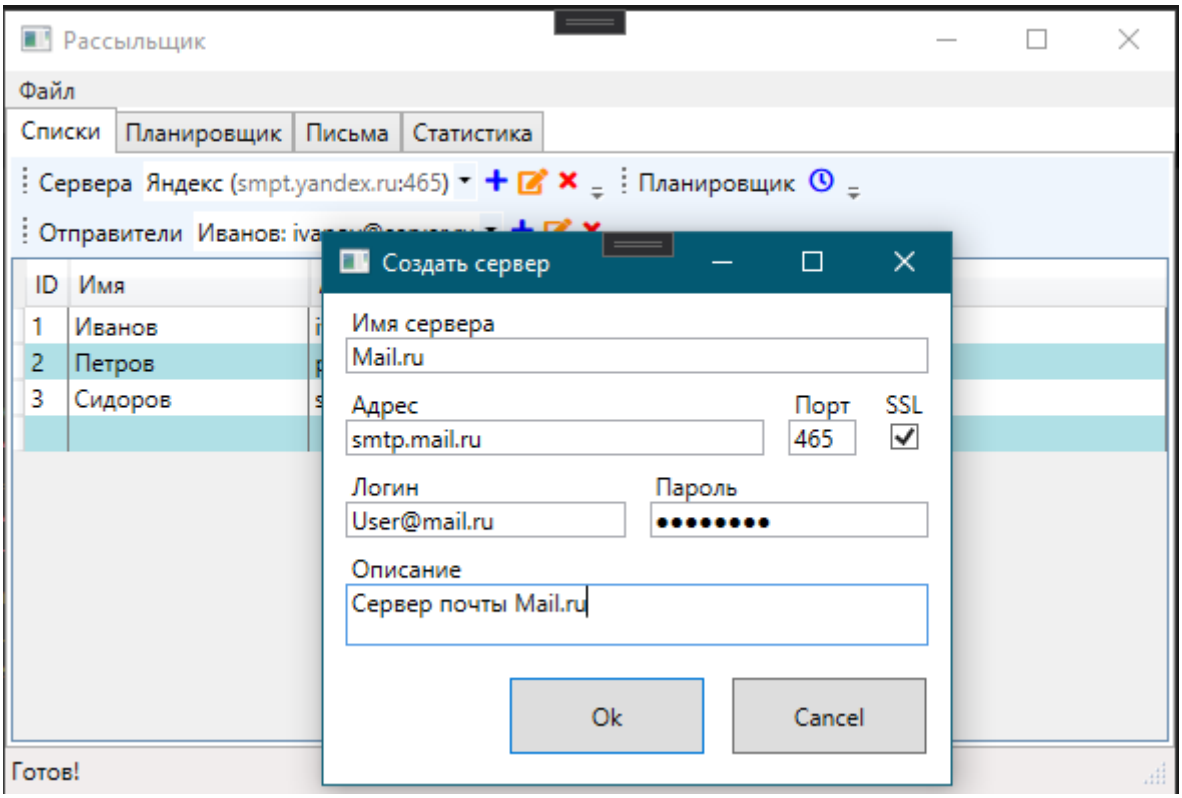
Код логики Главного окна

```
using System.Diagnostics;
using System.Linq;
using System.Net.Mail;
using System.Windows;
using MailSender.Data;
using MailSender.Models;
using MailSender.Services;

namespace MailSender
{
    public partial class MainWindow
    {
        public MainWindow() => InitializeComponent();

        /// <summary>Обработчик события кнопки создания нового сервера</summary>
        private void OnAddServerButtonClick(object Sender, RoutedEventArgs E)
        {
            if (!ServerEditDialog.Create(
                out var name,
                out var address,
                out var port,
                out var ssl,
                out var description,
                out var login,
                out var password))
                return;

            var server = new Server
            {
                Id = TestData.Servers.DefaultIfEmpty().Max(s => s.Id) + 1,
                Name = name,
                Address = address,
                Port = port,
                UseSSL = ssl,
                Description = description,
                Login = login,
```

```
                Password = password
            };

            TestData.Servers.Add(server);

            ServersList.ItemsSource = null;
            ServersList.ItemsSource = TestData.Servers;
            ServersList.SelectedItem = server;
        }

        /// <summary>Обработчик события кнопки редактирования сервера</summary>
        private void OnEditServerButtonClick(object Sender, RoutedEventArgs E)
        {
            if (!(ServersList.SelectedItem is Server server)) return;

            var name = server.Name;
            var address = server.Address;
            var port = server.Port;
            var ssl = server.UseSSL;
            var description = server.Description;
            var login = server.Login;
            var password = server.Password;

            if (!ServerEditDialog.ShowDialog("Редактирование сервера",
                ref name,
                ref address, ref port, ref ssl,
                ref description,
                ref login, ref password))
                return;

            server.Name = name;
            server.Address = address;
            server.Port = port;
            server.UseSSL = ssl;
            server.Description = description;
            server.Login = login;
            server.Password = password;

            ServersList.ItemsSource = null;
            ServersList.ItemsSource = TestData.Servers;
        }

        /// <summary>Обработчик события кнопки удаления сервера</summary>
        private void OnDeleteServerButtonClick(object Sender, RoutedEventArgs E)
        {
            if (!(ServersList.SelectedItem is Server server)) return;

            TestData.Servers.Remove(server);

            ServersList.ItemsSource = null;
            ServersList.ItemsSource = TestData.Servers;
            ServersList.SelectedItem = TestData.Servers.FirstOrDefault();
        }
```

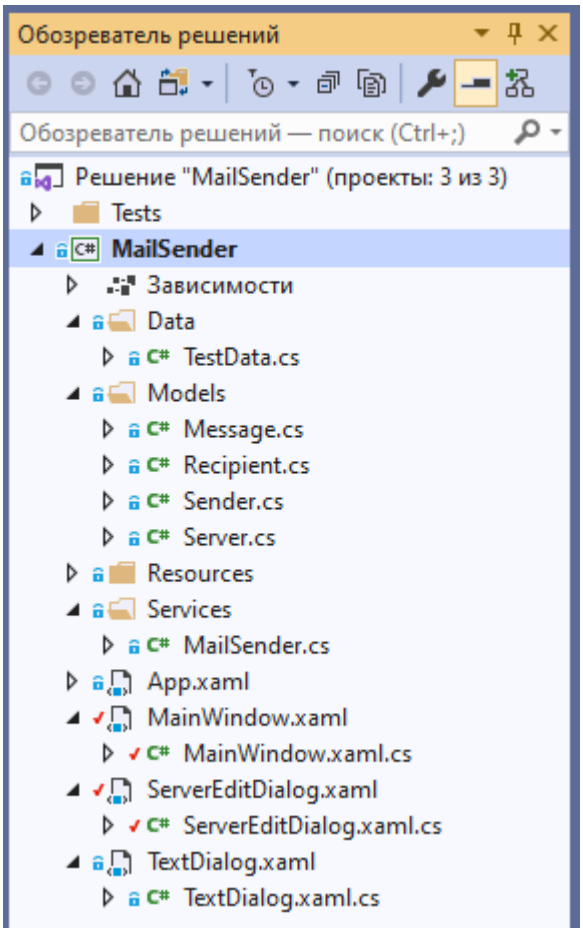
```
        /// <summary>Обработчик события кнопки немедленной отправки поты</summary>
        private void OnSendNowButtonClick(object Sender, RoutedEventArgs E)
        {
            // Извлекаем исходные параметры по возможности
            if (!(SendersList.SelectedItem    is Sender    sender))    return;
            if (!(RecipientsList.SelectedItem is Recipient recipient)) return;
            if (!(ServersList.SelectedItem    is Server    server))    return;
            if (!(MessagesList.SelectedItem   is Message   message))   return;
            // Если одни из параметров невозможно получить, то выходим

            // Создаём объект-рассыльщик и заполняем параметры сервера
            var mail_sender = new SmtpSender(
                server.Address, server.Port, server.UseSSL,
                server.Login, server.Password);

            // При отправке почты может возникнуть проблема.
            // Ставим перехват исключения.
            try
            {
                // Запускаем таймер
                var timer = Stopwatch.StartNew();
                // И запускаем процесс отправки почты
                mail_sender.Send(
                    sender.Address, recipient.Address,
                    message.Tittle, message.Body);
                timer.Stop(); // По завершении останавливаем таймер

                // Если почта успешно отправлена, то отображаем диалоговое окно
                var elapsed = timer.Elapsed.TotalSeconds;
                MessageBox.Show(
                    $"Почта успешно отправлена за {elapsed:0.##}с",
                    "Отправка почты",
                    MessageBoxButton.OK,
                    MessageBoxImage.Information);
            }
            // Если случилась ошибка, то перехватываем исключение
            catch (SmtpException) // Перехватывает строго нужное нам исключение!
            {
                MessageBox.Show(
                    "Ошибка при отправке почты",
                    "Отправка почты",
                    MessageBoxButton.OK,
                    MessageBoxImage.Error);
            }
        }
    }
}
```

Внешне полученные окна выглядят следующим образом:



Библиотека классов

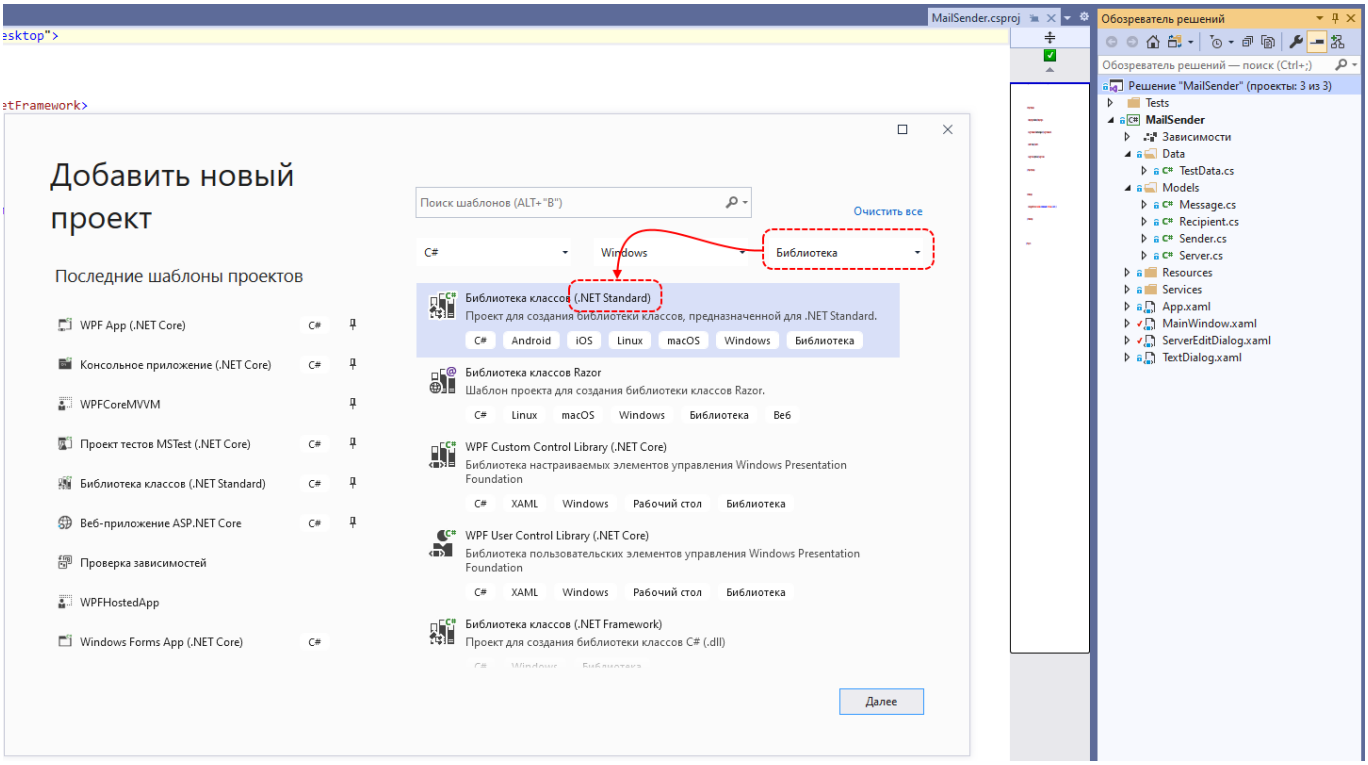
На текущий момент вся логика, модели данных и разметка у нас сосредоточены в одном месте - в основном проекте.



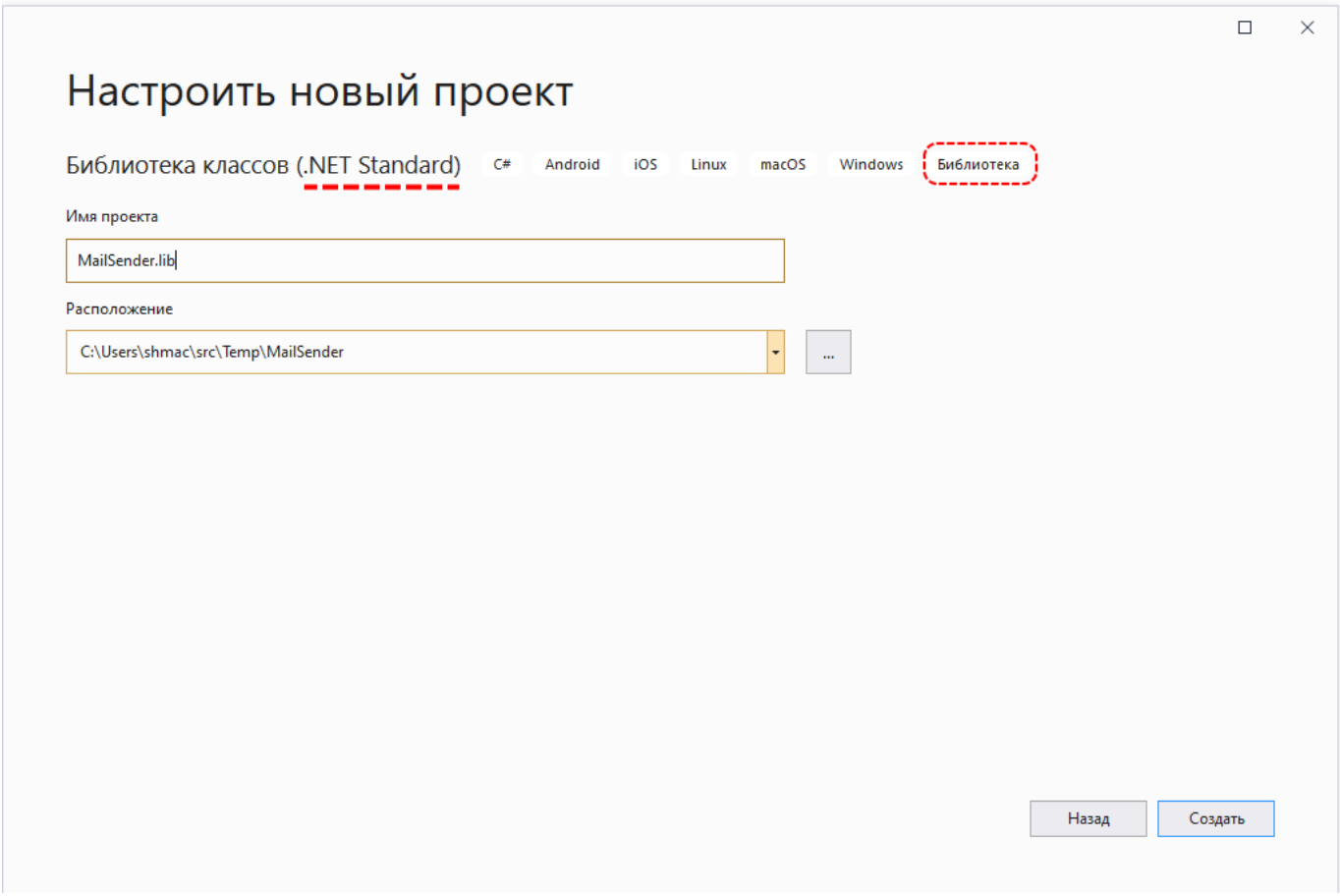
Если требуется создавать, поддерживать, развивать большой проект, либо работать в команде из нескольких разработчиков, либо имеется потенциальная возможность использования кода проекта в других проектах, то имеет смысл выполнить максимально возможное деление проекта на модули с чёткой специализацией. Каждый модель (сборка в определении .NET) может быть как исполнительным файлом (`.exe`), так и файлом библиотеки (`.dll`). Модуль библиотеки отличается от исполнительного файла лишь наличием точки входа (метод `Main`).

Создадим новый проект библиотеки классов (формата `.NET Standard` как наиболее унифицированного и кроссплатформенного) и выделим туда всю логику моделей и сервисов. Назовём проект `MailSender.lib`.

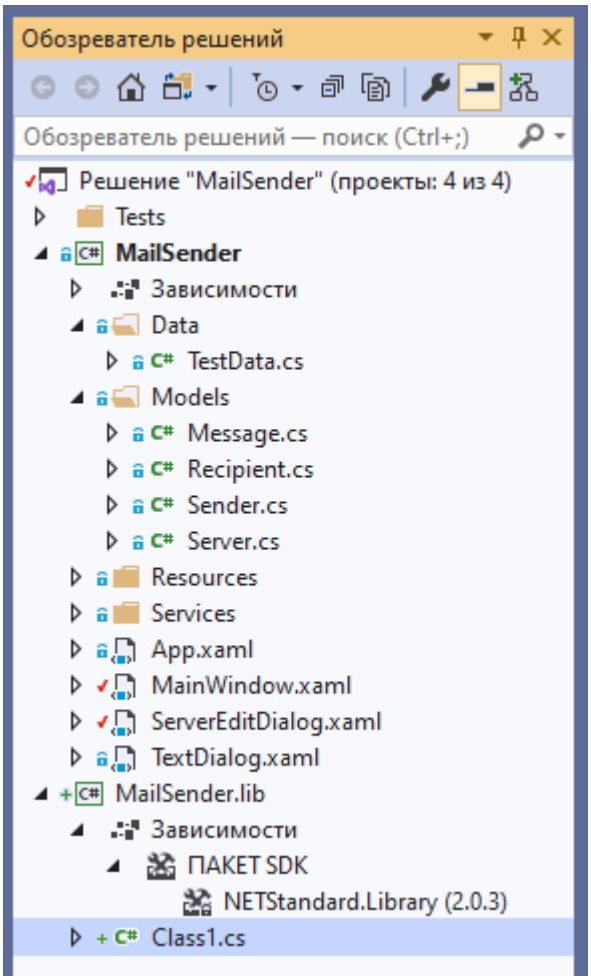
Кликаем правой кнопкой мыши по узлу решения в Обозревателе решения и выбираем "Добавить/Новый проект"



В диалоге выбора шаблона проекта нас интересует раздел C#/Windows/Библиотека

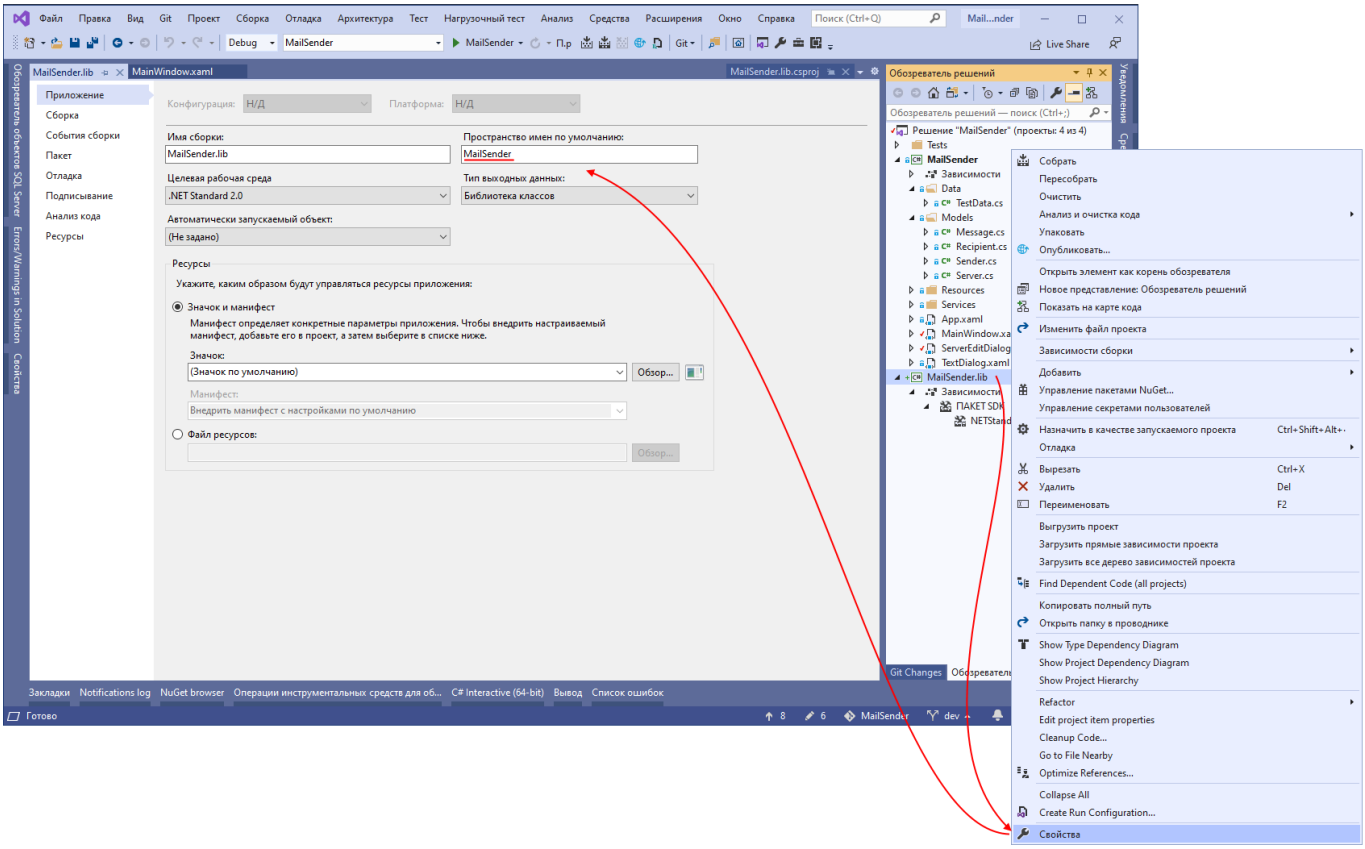


И выбираем проект библиотеки `.NET Standard`.



В результате мы получи новый проект, "лежащий" рядом с основным проектом.

Изменим пространство имён этого проекта по умолчанию на наше основное пространство имён `MailSender`. Для этого в Обозревателе решений надо кликнуть правой кнопочкой мыши и вызвать контекстное меню у созданного проекта и выбрать в нём пункт "Свойства".

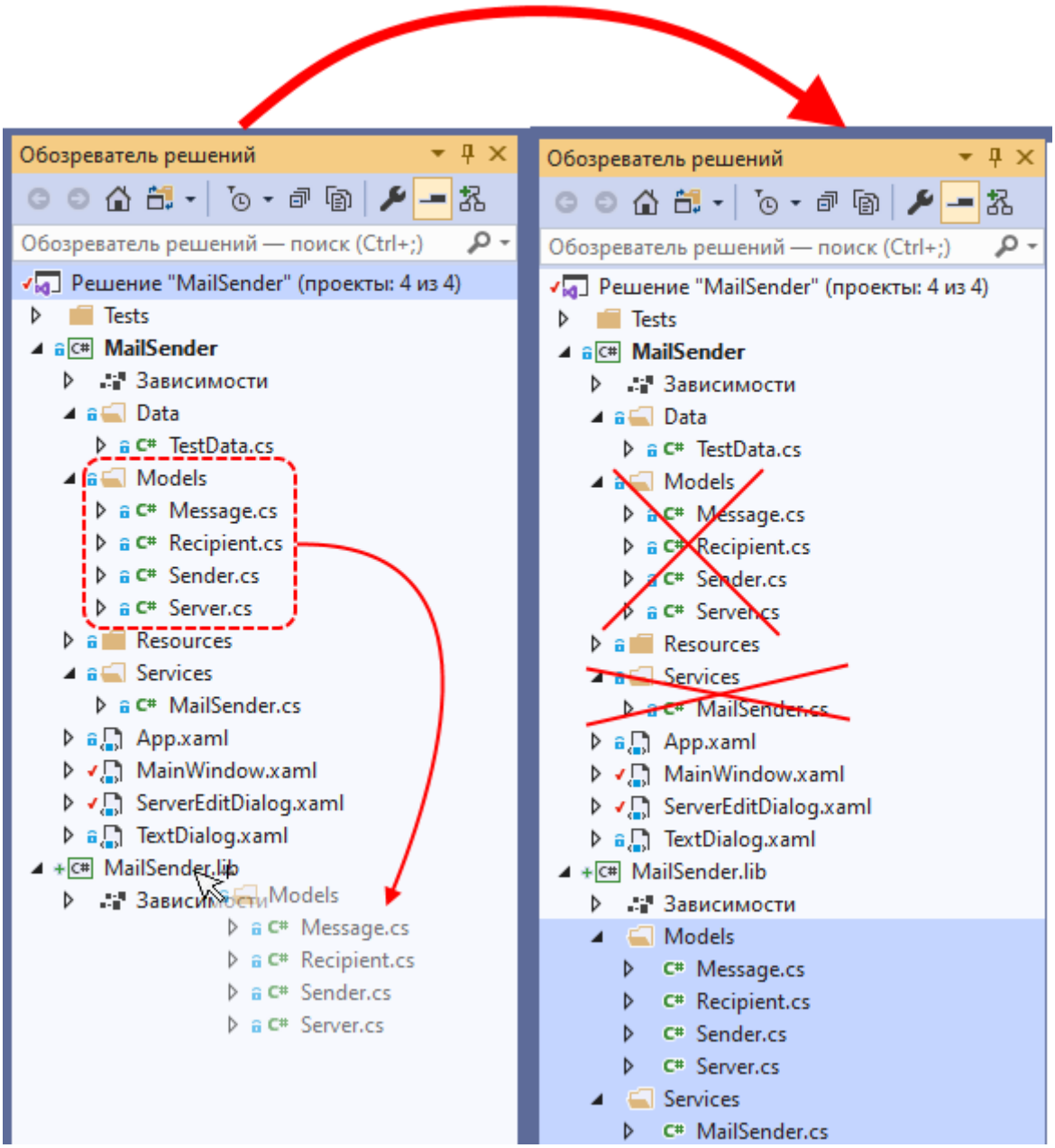


В открывшемся диалоге свойств проекта меняем пространство имён на нужное нам.

Это же можно сделать вручную отредактировав файл проекта `MailSender.lib.csproj`

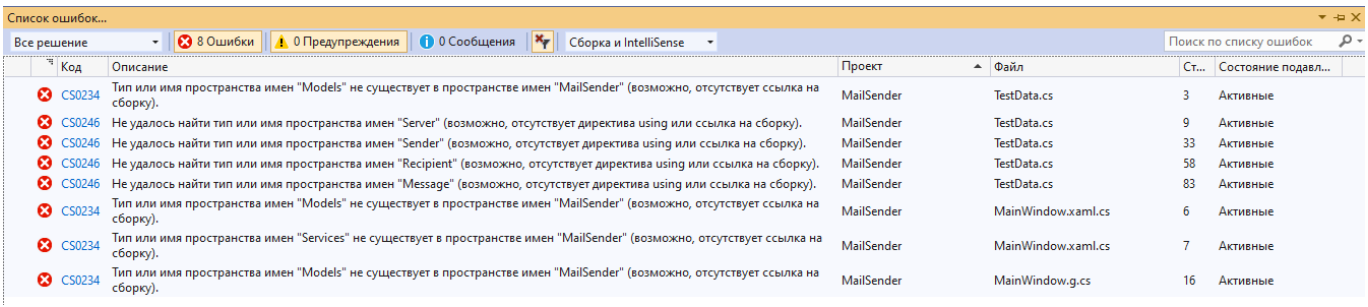
```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
    <RootNamespace>MailSender</RootNamespace> <!-- можно добавить вручную -->
  </PropertyGroup>

</Project>
```

Теперь перенесём в этот проект мышкой файлы моделей и сервисов из основного проекта

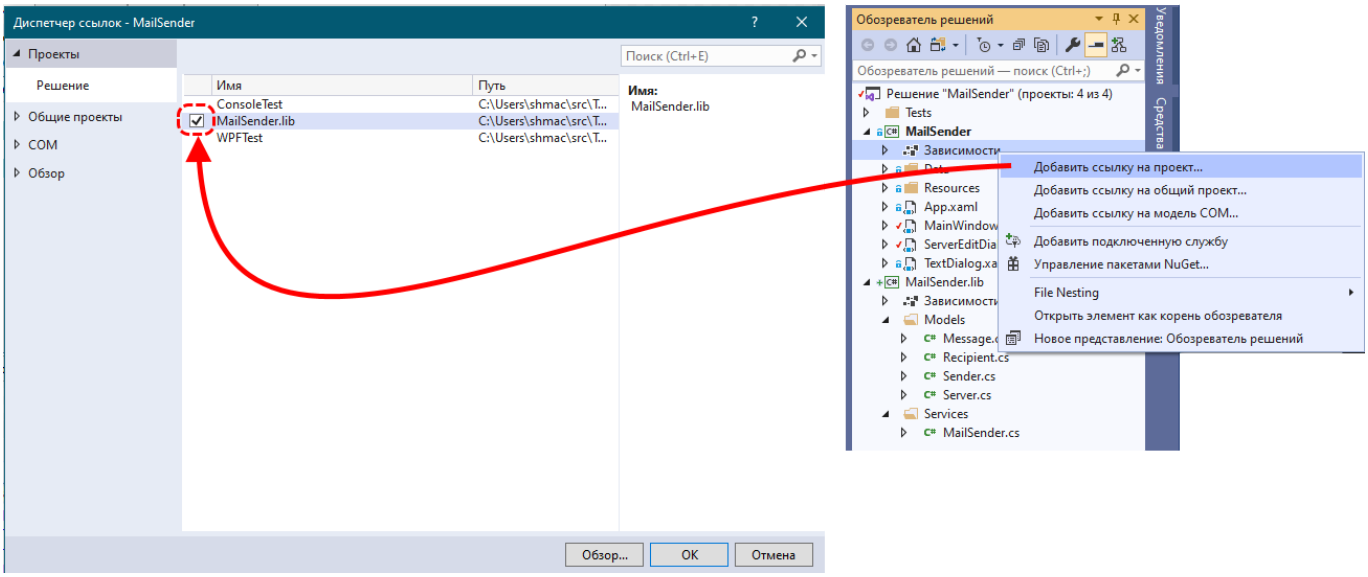


А в основном проекте удалим эти файлы.

При попытке собрать проект компилятор законно будет нами недоволен:



Мы удалили код из основного проекта и в результате, а ссылки на него у нас остались. В результате компилятор не понимает что мы хотели этим сказать. Давайте ему поможем: укажем в основном проекте ссылку на проект библиотеки:



Этого же можно добиться, отредактировав вручную файл основного проекта `MailSender.csproj`

```
<Project Sdk="Microsoft.NET.Sdk.WindowsDesktop">

  <PropertyGroup>
    <!-- Просим компиялтор сделать нам файл .exe -->
    <OutputType>WinExe</OutputType>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <!-- Указываем, что сборка должна использовать WPF -->
    <UseWPF>true</UseWPF>
    <!-- Версия C# самая последняя -->
    <LangVersion>preview</LangVersion>
  </PropertyGroup>

  <ItemGroup>
    <!-- Ссылка на пакет NuGet -->
    <PackageReference Include="FontAwesome5" Version="2.0.8" />
  </ItemGroup>

  <ItemGroup>
    <!-- Ссылка на проект -->
    <ProjectReference Include="..\MailSender.lib\MailSender.lib.csproj" />
  </ItemGroup>

</Project>
```

И точно также "поднимем" версию C# до последней в проекте библиотеки:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
    <RootNamespace>MailSender</RootNamespace>
    <!-- Версия C# самая последняя -->
```

```
    <LangVersion>latest</LangVersion>
  </PropertyGroup>

</Project>
```

Также надо сделать, чтобы "переезд" кода в отдельную сборку перестал приводить к ошибкам компиляции - надо сделать классы видимыми (публичными). Для этого во всех классах в сборке, которыми Вы рассчитываете пользоваться во вне её, надо добавить (изменить) модификатор доступа на `public`

На пример для класса `Message`

```
public class Message
{
    public int Id { get; set; }
    public string Tittle { get; set; }
    public string Body { get; set; }
}
```

Ну и последняя ошибка в разметке главного окна. Мы объявляли псевдоним пространства имён `xmlns:m="clr-namespace:MailSender.Models"`, но отныне в главном проекте такого пространства имён нет. Оно находится в сборке `MailSender.lib`. Нам необходимо скорректировать это объявление указав имя сборки: `xmlns:m="clr-namespace:MailSender.Models;assembly=MailSender.lib"`

Теперь проект компилируется и запускается, а в выходной папке `MailSender\MailSender\bin\Debug\netcoreapp3.1` образуется помимо файла `MailSender.exe` и файла `MailSender.dll` ещё и файл `MailSender.lib.dll`.

Сборка представляет собой файл, в котором содержатся результаты компиляции проекта:

- промежуточный код общеязыковой среды исполнения (`CLR`), называемый IL-кодом
- ресурсы - строки, картинки, двоичные данные, упакованные компилятором
- метаинформация о коде - названия типов, методов, переменных, атрибуты и др.

При запуске приложения в память загружается лишь главная его сборка (основной исполнительный файл - в нашем случае `MailSender.exe`). Остальные сборки загружается в память по мере необходимости (по мере использования типов в коде программы). Как только в процессе выполнения программы речь доходит до одной из наших моделей, либо сервисов, платформа .NET пытается найти на диске и загрузить сборку, в котором этот тип определён. Поиск файла сборки осуществляется в следующем порядке:

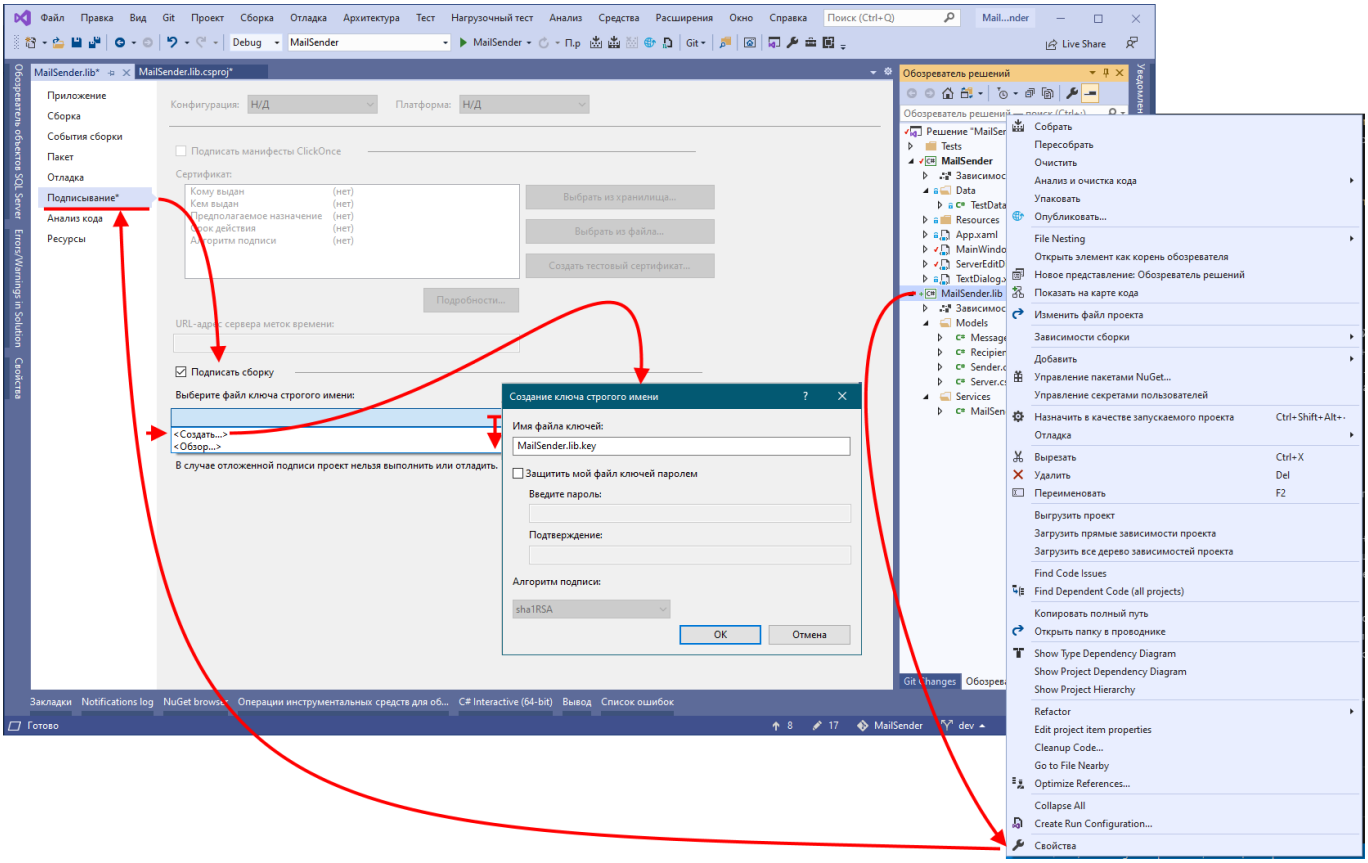
1. В директории расположения исполнительного файла
2. Во всех её поддиректориях
3. В GAC (Global Assembly Cache) - глобальном кэше сборок.

Кеш сборок - это унифицированное место расположения библиотек (сборок .NET) в операционной системе, доступных для всех приложений, запускаемых в этой ОС.

Когда-то, с появлением понятия динамически подключаемой библиотеки, а также с ростом их популярности и количества, появилась и проблема, известная как "DLL hell" ("ад сборок"). Его смысл в каше, Которая возникает в общедоступном месте расположения библиотек, когда каждое приложение начинает добавлять туда свои файлы. Два приложения могут в своём составе использовать одну и ту же библиотеку, но с разной версией. А имя у файла будет одно и то же.

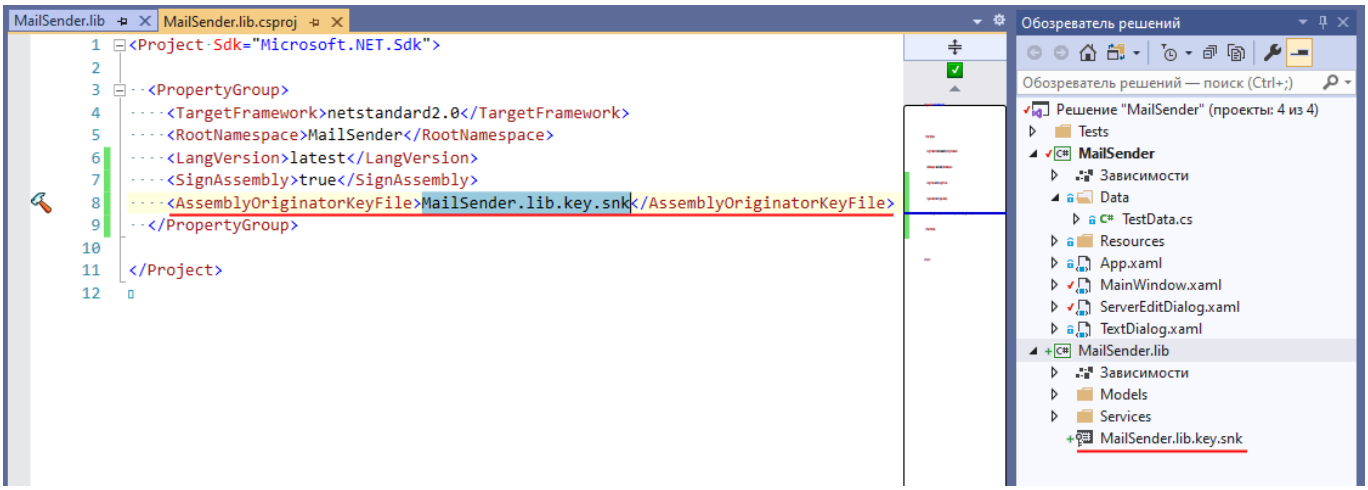
Данная проблема в GAC решается за счёт применения "строгого имени" сборки. Каждая сборка идентифицируется своим именем. Имя сборки может быть "слабым", включающим в себя лишь имя файла, и сильным. Сильное имя помимо имени файла, включает в себя ещё, версию и цифровую подпись (контрольную сумму). Для размещения сборки в GAC необходимо "подписать сборку строгим именем". Давайте проделаем это для нашей новой библиотеки логики.

Для этого в обозревателе решений надо найти узел проекта библиотеки и в контекстном меню выбрать "Свойства". В открывшемся дизайнере свойств надо перейти на страницу "Подписывание".



Выбираем имя файла и можем указать пароль. Если пароль не указать, то любой сможет декомпилировать сборку, исправить её и заново подписать её той же цифровой подписью. Это делать имеет смысл если ваш проект является проектом с открытым исходным кодом, который Вы готовы позволить исправлять всем под свои нужды. Для коммерческих проектов либо будет выдан готовый файл ключа, либо следует сгенерировать файл ключа защищённый паролем.

В результате в проект будет добавлен файл ключа



Код файла проекта в этом случае указывает необходимость операции подписывания, которую должен будет выполнить компилятор на финальной стадии компиляции.

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
    <RootNamespace>MailSender</RootNamespace>
    <LangVersion>latest</LangVersion>
    <SignAssembly>true</SignAssembly>

    <!-- Подписывание сборки - указывается файл ключа -->
    <AssemblyOriginatorKeyFile>MailSender.lib.key.snk</AssemblyOriginatorKeyFile>
  </PropertyGroup>

</Project>
```

Физически GAC размещается в `C:\Windows\Microsoft.NET\assembly\GAC_MSIL` Для того чтобы добавить сборку `.NET Framework` в GAC надо выполнить команду `C:\Program Files\Microsoft SDKs\Windows\v8.0A\bin\NETFX 4.0 Tools>gacutil -i Полное_имя_сборки`

В .NET Core вместо GAC используется концепция "Хранилища пакетов среды выполнения".

Домашнее задание

1. Создать разметку интерфейса основного приложения для
    1. Управления списками рассылки/серверов/отправителей
    2. Управления планировщиком заданий
    3. Управления списком писем
    4. Просмотра статистики
2. Создать основу логики приложения
    1. Набор классов-моделей
    2. Класс сервиса отправки почты
3. Интегрировать логику и интерфейс обеспечив:
    1. Управление списками (добавление/удаление/редактирование) элементов списков

    2. Отправку почты
    3. Просмотр результатов процесса отправки почты
4. Выделить логику приложения в отдельную библиотеку и использовать её в основном приложении добавив ссылку на библиотеку
5.  По средствам менеджера пакетов добавить в проект основного приложения NuGet-пакет WPF ToolKit версии 3.8.2 (последняя нелицензируемая версия) и использовать из этой библиотеки элемент для выбора времени чтобы создать логику формирования заданий планировщика.

Ссылки

- XAML - общие сведения
- Панели компоновки
- Элементы управления WPF
- Сборки .NET
- GAC - Глобальный кэш сборок
- Хранилища пакетов .NET Core
- Менеджер пакетов NuGet
- Подписывание сборок строгим именем
- CLR - общеязыковая среда исполнения
- IL-код
- Полезные пакеты NuGet
    - WPFToolKit - полезные элементы управления (версия ниже 4.x не требует лицензирования)
    - WontAwesome.WPF - пакет со значками из FontAwesome