

АННОТАЦИЯ

Отчет 122 с., 6 ч., 99 рис., 55 табл., 14 источников, 10 прил.

ВЕБ-ТЕХНОЛОГИИ, HTML, CSS, SASS, JAVASCRIPT, TYPESCRIPT, BLAZOR, GULP, TELMA, WPF, MVVM, ИНТЕРФЕЙС, ОПТИМИЗАЦИЯ.

Объект исследования — функциональные возможности технологии Blazor для создания интерфейсов веб-приложений.

Цель работы — исследование возможности создания веб-версии пользовательского интерфейса для WPF приложения на основе технологии Blazor.

В ходе выполнения работы были рассмотрены различные веб-технологии и их использование для создания веб-интерфейсов. Также была создана подробная документация в виде каталога реализованных компонентов. Дополнительно были рассмотрены способы оптимизации процесса разработки веб-приложения и его производительности.

В результате выполнения работы был реализован полноценный веб-интерфейс двумерной части программного комплекса Telma, с помощью которого можно выполнять задание и решение двумерных физических задач. Для элементов управления, созданных с помощью WPF и XAML, в веб-интерфейсе были реализованы соответствующие аналоги.

СОДЕРЖАНИЕ

Введение.....	6
1. Используемые технологии	9
1.1. Архитектура MVVM.....	9
1.2. Описание используемых веб-технологий.....	10
2. Использование технологии Blazor для создания MVVM View.....	15
2.1. Построение страниц веб-приложения.....	15
2.2. Создание компонентов	18
2.3. Взаимодействие с JavaScript из Blazor.....	21
3. Каталог элементов управления пользовательского интерфейса.....	25
3.1. Общие правила создания компонентов	26
3.2. Элементы управления для ввода и выбора данных.....	30
3.2.1. Поля для текста и числовых значений.....	30
3.2.2. Выпадающие списки для выбора значения.....	40
3.2.3. Элементы управления для выбора значений	44
3.3. Элементы управления для отображения данных.....	48
3.3.1. Элементы для вывода контента.....	48
3.3.2. Элементы со скрытым контентом	52
3.4. Кнопки.....	55
3.5. Компоненты для построения страниц.....	62
3.5.1. Панели	62
3.5.2. Окна для взаимодействия с пользователем.....	67
3.6. Обертки для элементов интерфейса.....	69
3.7. Упрощение и оптимизация процесса разработки	72

4. Реализованный функционал MVVM View	74
4.1. Внешний вид страниц приложения	74
4.2. Панели в верхней части экрана	77
4.3. Панели в боковой части экрана	78
4.4. Всплывающие окна и списки	85
4.5. Окна для задания параметров в препроцессоре	88
4.6. Интерактивные возможности интерфейса	95
5. Оптимизация работы интерфейса пользователя	96
5.1. Критические этапы рендеринга	96
5.2. Сокращение размера файлов приложения	97
5.3. Оптимизация в Google PageSpeed Insights	99
6. Тестирование View с помощью расчета конечноэлементной задачи	104
Заключение	107
Список литературы	108
Приложение А. Текст программы для целочисленного поля IntInput	110
Приложение Б. Текст программы для DictionarySelect	112
Приложение В. Текст программы для элемента выбора Checkbox	113
Приложение Г. Текст программы для элемента вывода списка List	114
Приложение Д. Текст программы для блока Expander	115
Приложение Е. Текст программы для кнопки с картинкой ImageButton	116
Приложение Ж. Текст программы для кнопки с текстом TextButton	118
Приложение З. Текст программы для работы с вкладками TabControl	120
Приложение И. Текст программы для отображения вкладок TabPage	121
Приложение К. Текст программы для окон параметров Parameters	122

ВВЕДЕНИЕ

В последние годы активно набирают популярность веб-технологии. Они широко используются при разработке программного обеспечения благодаря большому набору инструментов, позволяющих создавать гибкие адаптивные пользовательские интерфейсы любой степени сложности и интерактивности. Другим преимуществом веб-технологий является их высокая скорость работы и производительность. Также важным достоинством веб-технологий является их доступность и кроссплатформенность, т.е. способность программного обеспечения работать на различных устройствах, платформах и операционных системах. Кроссплатформенность в данном случае достигается благодаря тому, что доступ к веб-приложению может быть получен с любого устройства, имеющего браузер и доступ в интернет.

Веб-технологии находят применение в различных областях человеческой деятельности, начиная с создания сайтов по любой тематике и заканчивая созданием крупных веб-приложений, сервисов и программных комплексов. Ранее, начиная со второго курса обучения, в рамках проектной деятельности по различным дисциплинам рассматривались возможности применения веб-технологий при разработке программного обеспечения.

В результате проделанной ранее работы с помощью веб-технологий была создана виртуальная лаборатория с трехмерной динамической визуализацией прецессии и нутации гироскопа в реальном времени [1, 7]. Данная разработка используется для демонстрации теоретического материала на лекционных занятиях по физике, а также может применяться при проведении лабораторных работ с необходимым оборудованием в дистанционном формате.

Кроме того, с помощью веб-технологий также было создано интерактивное электронное учебное пособие по математическому анализу, используемое на лекционных, практических и семинарских занятиях у студентов различных курсов, а также для их самостоятельной работы без присутствия преподавателя [3].

Результаты проделанной работы были отмечены стипендией мэрии города Новосибирска за научную деятельность, государственной академической стипендией за достижения в научно-исследовательской деятельности, грантом Новосибирского государственного технического университета для выполнения научной работы и стипендией компании Huawei. Также полученные результаты были отмечены дипломами первой степени за лучшие доклады на Международной научно-практической конференции «Электронные средства и системы управления» и на Всероссийской научной конференции молодых ученых «Наука. Технологии. Инновации».

Результаты работы над виртуальной лабораторией и электронным учебным пособием показали высокую эффективность применения веб-технологий для разработки программного обеспечения, а также актуальность, значимость и перспективность данного направления исследований. Продолжением работы в данной области является исследование возможности создания веб-версии пользовательского интерфейса для WPF приложения на основе технологии Blazor.

В качестве WPF приложения, на основе которого будет создана веб-версия MVVM View, была выбрана двумерная часть программного комплекса Telma. Данный комплекс предназначен для численного решения тепловых и электромагнитных задач различной сложности с помощью метода конечных элементов (МКЭ).

Метод конечных элементов широко используется многими исследователями при работе с математическими моделями физических процессов в виде краевых и начально-краевых задач для дифференциальных уравнений с частными производными [4, с. 14]. Использование этого метода позволяет заменить задачу поиска некоторой неизвестной функции на задачу поиска набора ее приближенных значений в узлах конечных элементов. При этом дифференциальное уравнение, задающее закон изменения искомой функции, заменяется системой линейных алгебраических уравнений (СЛАУ), которая может быть решена одним из известных прямых или итерационных методов.

При решении задач с помощью МКЭ производится конечноэлементная дискретизация расчетной области. В качестве конечных элементов в двумерном случае могут использоваться треугольники, четырехугольники или шестиугольники, а в трехмерном случае — тетраэдры, параллелепипеды или призмы. На каждом конечном элементе выбираются финитные базисные функции, равные нулю вне этого элемента. Далее с помощью вариационной постановки осуществляется переход от дифференциального уравнения к совокупности интегралов для матриц жесткости и масс. После этого собирается матрица СЛАУ и вектор правой части, а также накладываются граничные условия. Результатом решения СЛАУ является некоторый вектор весов, задающий конечноэлементное представление искомой функции.

Программный комплекс Telma позволяет автоматизировать основные этапы решения физических задач методом конечных элементов. С его помощью можно решать линейные и нелинейные задачи в различных постановках, а также задачи с аппроксимацией по времени. В качестве конечных элементов для двумерных задач обычно используются треугольники, а для трехмерных — тетраэдры. В качестве базисных функций используются полиномы до третьего порядка включительно. Тип полиномов выбирается между эрмитовыми, иерархическими, лагранжевыми и некоторыми другими.

Telma состоит из трех модулей: препроцессора, процессора и постпроцессора. Препроцессор позволяет строить конечноэлементную сетку и добавлять материалы и краевые условия, в процессоре задаются параметры задачи, а в постпроцессоре выводится полученное распределение поля в цветовой градации или с помощью изолиний.

В данной работе реализуется пользовательский веб-интерфейс комплекса Telma для препроцессора, процессора и постпроцессора. Решение двумерных задач с использованием веб-интерфейса должно выполняться аналогично решению в десктопной версии приложения, созданной с помощью WPF. Для всех основных элементов управления Telma создаются соответствующие аналоги с использованием веб-технологий.

1. ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

1.1. АРХИТЕКТУРА MVVM

Model-View-ViewModel (MVVM) — паттерн проектирования архитектуры приложений, который применяется для разделения модели и ее представления для их самостоятельного и независимого существования и функционирования [14]. Использование этого паттерна позволяет разработчикам создавать графический пользовательский интерфейс и реализовывать логику взаимодействия с данными независимо друг от друга. MVVM был создан для упрощения событийно-ориентированного программирования приложений с помощью механизма привязки данных. Шаблон MVVM состоит из трех основных частей.

Модель (Model) представляет собой описание данных, необходимых для работы программного обеспечения, а также логику работы с этими данными.

Представление (View) — это графический интерфейс пользователя, включающий в себя кнопки, списки, таблицы, текстовые поля и другие элементы управления. View обрабатывает взаимодействие пользователей с интерфейсом, такие как щелчки мыши, касания экрана или ввод с клавиатуры. Представление подписывается на изменение состояния свойств, полей или команд, которые предоставляются Моделью представления (ViewModel). Если во ViewModel изменилось какое-либо свойство, View запрашивает у ViewModel значение обновленного свойства. Если же пользователь совершает действие с помощью одного из элементов интерфейса, View запрашивает у ViewModel выполнение команды, соответствующей совершенному действию. Такая связь View и ViewModel осуществляется с помощью механизма привязки данных.

Модель представления (ViewModel) сообщает всем подписчикам об изменении какого-либо свойства в Model или о совершении какого-либо действия пользователя во View для отправки и получения обновлений. ViewModel содержит команды, с помощью которых View может влиять на Model, а также свойства Model, которые отображает View.

В шаблоне MVVM организовано строгое взаимодействие между составляющими в виде двунаправленной связи Model с ViewModel и View с ViewModel. При этом View не может изменять Model напрямую, а может влиять на нее только через ViewModel. В этом заключается основное отличие MVVM от паттернов MVC (Model-View-Controller) и MVP (Model-View-Presenter). При использовании MVC каждая из составляющих может влиять на другую, что приводит к трудоемкому отслеживанию направления управляющих потоков. Шаблон MVP был порожден из MVC и отличается от него наличием односторонней связи между Model и Presenter.

1.2. ОПИСАНИЕ ИСПОЛЬЗУЕМЫХ ВЕБ-ТЕХНОЛОГИЙ

Для разработки MVVM View на основе программного комплекса Telma использовались несколько основных веб-технологий, среди которых HTML, CSS, JavaScript и Blazor. Рассмотрим данные технологии и их приложение к созданию графического пользовательского интерфейса.

HTML (HyperText Markup Language) — язык гипертекстовой разметки и форматирования веб-документов в сети Интернет, который позволяет создавать и описывать структуру веб-страниц и содержащийся в них контент [2]. Наиболее часто при разработке используется HTML пятой версии. Спецификация, основные принципы и стандарты описания документов посредством языка HTML определяются Консорциумом Всемирной паутины (World Wide Web Consortium, W3C). Любой веб-документ состоит из набора семантических тегов, определяющих назначение и смысл того или иного структурного блока страницы. Теги создают иерархию блоков страницы и структурируют ее содержимое. Различные HTML-теги имеют разные функции, атрибуты, свойства и внешний вид. Элементы страницы могут быть вложены друг в друга, а также содержать или не содержать контент. Браузер интерпретирует HTML в процессе загрузки страницы и формирует форматированный текст, который в конечном итоге отображается на экране пользователя в виде различных элементов.

Для описания внешнего вида веб-страниц, написанных с помощью HTML, используется формальный язык CSS (Cascading Style Sheets). CSS предоставляет большой набор инструментов для изменения различных аспектов визуального представления элементов пользовательского интерфейса [6]. С его помощью можно изменять шрифты, цвета, размеры и поведение элементов страницы, их положение относительно других элементов, а также их форму, видимость, прозрачность, наложение и многое другое. Кроме того, язык CSS позволяет изменять внешний вид интерфейса в зависимости от размеров экрана и типов устройств, на которых отображается приложение. Такой широкий спектр возможностей для визуального оформления веб-страницы позволяет создавать пользовательские интерфейсы любой степени сложности и интерактивности.

Использование CSS позволяет разделить описание внешнего вида веб-страницы и ее логической структуры, создаваемой с помощью языков разметки. С помощью такого разделения можно обеспечить большую гибкость при работе с веб-документом, улучшить его доступность и ускорить процесс исправления, доработки и масштабирования приложений.

Стилизация элементов веб-страницы выполняется с помощью задания специальных CSS-правил соответствующим HTML-тегам. Для этого каждому тегу в HTML присваивается идентификатор, с помощью которого этот тег может быть найден в разметке страницы. Существует два основных идентификатора: `id` и `class`, при этом `id` должен быть уникален в пределах документа и соответствовать только одному HTML-тегу, а идентификатор `class` может быть присвоен многим тегам. Также каждый элемент может иметь несколько классов, определяющих его различные свойства, но должен иметь лишь один `id`. Для написания CSS-правила необходимо указать селектор — один из идентификаторов HTML-тега, его атрибут или непосредственно сам тег, а также создать блок объявлений, в котором будет находиться одно или более CSS-объявление. Каждое CSS-объявление представляет собой сочетание CSS-свойства и значения и задает один из аспектов внешнего вида или поведения элемента веб-страницы.

Если для создания структуры страницы используется HTML, а для ее визуального оформления используется CSS, то обеспечение функциональности и интерактивности страницы реализуется с помощью языка программирования JavaScript [5]. JavaScript широко используется в веб-разработке в качестве интерпретируемого языка сценариев для управления HTML-элементами страницы и описания их поведения, а также для обработки действий пользователя и обеспечения интерактивности веб-страниц на стороне клиента.

Язык JavaScript имеет большое количество возможностей для управления объектной моделью документа (Document Object Model, DOM) — интерфейсом программирования приложений для документов, написанных с использованием HTML. Согласно концепции DOM, веб-документ может быть представлен как дерево объектов, узлы которого представляют собой элементы страницы. DOM-дерево отражает иерархию между HTML-тегами и связи между ними. При этом одни узлы дерева могут быть родительскими по отношению к другим узлам. Благодаря такому представлению веб-страниц, с помощью JavaScript можно удобно манипулировать узлами DOM-дерева. В частности, JavaScript позволяет получать, изменять, копировать, добавлять и удалять узлы, а также переносить их в пределах DOM-дерева и изменять связи между ними.

Работа с объектной моделью документа осуществляется с помощью создания пользовательских скриптов (сценариев) на языке JavaScript. Скриптом называется определенная последовательность команд, описанных с использованием скриптового языка программирования для автоматизированного выполнения набора задач, обеспечивающих некоторую функциональность. Использование скриптов позволяет организовать управление элементами веб-страницы и задать им необходимое поведение и отображение. В качестве примеров работы скриптов в веб-приложениях можно привести автоматическое перестроение или переформатирование страницы, автозаполнение форм, появление и скрытие всплывающих окон и многое другое. Все пользовательские скрипты выполняются в браузере пользователя при загрузке веб-страницы.

Разработка функциональности веб-приложения с использованием одного лишь JavaScript может быть затруднительной или недостаточно эффективной в силу некоторых ограничений самого языка программирования JavaScript. Например, реализация объектно-ориентированного подхода в JavaScript основана не на классах, а на прототипах, что усложняет проектирование архитектуры приложений по сравнению с использованием других языков программирования, реализующих объектно-ориентированную парадигму на основе классов [8]. Другим немаловажным фактором является то, что при разработке пользовательского интерфейса возникает необходимость создания элементов управления, работающих со строго определенными типами данных. Для этой цели хорошо подходят языки со строгой статической типизацией, а язык JavaScript имеет динамическую типизацию. Существует несколько решений данных проблем, основанных на совместном использовании JavaScript и другого языка, обладающего описанными ранее свойствами. Рассмотрим два наиболее эффективных и широко используемых решения.

Первое решение заключается в использовании языка программирования TypeScript, расширяющего возможности JavaScript путем введения явной статической типизации и поддержки использования полноценных классов для реализации объектно-ориентированного подхода. Кроме того, TypeScript поддерживает возможность подключения программных модулей, что позволяет повысить читаемость кода, облегчить его рефакторинг и повторное использование, а также ускорить процесс разработки, поиск ошибок и их устранение [9].

Поскольку язык TypeScript является надстройкой над JavaScript, то он компилируется в JavaScript и обладает полной обратной совместимостью с ним. Отсюда следует, что программа, написанная на JavaScript, может быть переведена на TypeScript, а программа на TypeScript может включать JavaScript и использовать его библиотеки. Благодаря этому перенос пользовательских скриптов с JavaScript на TypeScript при необходимости может осуществляться постепенно с помощью поэтапного последовательного определения типов.

Второе решение заключается в совместном использовании JavaScript и специальных фреймворков, позволяющих разрабатывать интерактивные веб-интерфейсы с использованием языка программирования C# [13]. Использование C# обусловлено отчасти поддержкой статического объявления типов и реализацией объектно-ориентированного подхода на основе классов по умолчанию. Примером такого фреймворка является веб-платформа Blazor [10].

Платформа Blazor позволяет реализовать функциональность интерфейса веб-приложения как с использованием только языка C#, так и при помощи взаимодействия с JavaScript на стороне клиента. Кроме того, Blazor предоставляет возможности для совместного использования клиентской и серверной логики приложения, а также для отображения графического интерфейса пользователя в виде страниц HTML с описанием их внешнего вида на CSS. Использование фреймворка Blazor позволяет получить высокий уровень безопасности, надежности и производительности .NET и делает возможным использование его стандартных библиотек.

Весь программный код на языке C#, который должен выполняться на клиенте, компилируется в WebAssembly и загружается вместе с другими файлами сценариев при открытии веб-страницы пользователем. WebAssembly поддерживается браузерами без подключения дополнительных библиотек и представляет собой специальный формат байт-кода, оптимизированный для повышения скорости загрузки и производительности веб-приложений.

Разработка View в данной работе требует взаимодействия с ViewModel из программного комплекса Telma, реализованного на языке C#. Исходя из этого, в качестве основной технологии для создания функциональности пользовательского интерфейса была выбрана платформа Blazor. При этом часть функционала и интерактивности View была вынесена в JavaScript по причине более простой, удобной и эффективной реализации в виде сценариев JavaScript. Также помимо Blazor для разработки некоторых функций, реализующих отрисовку двумерной и трехмерной графики в браузере, использовался язык TypeScript.

2. ИСПОЛЬЗОВАНИЕ ТЕХНОЛОГИИ BLAZOR ДЛЯ СОЗДАНИЯ MVVM VIEW

2.1. ПОСТРОЕНИЕ СТРАНИЦ ВЕБ-ПРИЛОЖЕНИЯ

Все страницы веб-приложения состоят из нескольких основных частей.

Документ начинается с декларации типа документа с помощью тега `<!DOCTYPE html>`. Этот атрибут необходим для того, чтобы браузер мог определять используемую версию HTML и корректно отображать загружаемую веб-страницу.

Далее записывается тег `<html>`, который является корневым элементом документа и включает в себя всю HTML-разметку. Для этого тега следует указывать атрибут `lang="en"`, явно указывающий английский язык в качестве основного языка документа.

После этого ставится тег `<head>`, который содержит необходимую техническую информацию о документе. В нем указываются специальные meta-теги, отвечающие за используемую кодировку, описание содержимого страницы, ключевые слова, совместимость с другими браузерами и некоторые другие параметры. Они используются для информирования браузера о том, как следует обрабатывать веб-страницу. Также внутри `<head>` используется тег `<title>`, задающий название страницы во вкладке браузера, и теги `<link>`, в которых указываются ссылки на внешние файлы CSS-стилей проекта, различные библиотеки, иконки, шрифты и другие ресурсы.

Завершает основную структуру HTML-документа тег `<body>`, который представляет собой тело документа и включает в себя содержимое страницы в виде html-тегов. Внутри `<body>` описывается вся разметка документа, после которой следует один или несколько тегов `<script>`, содержащих ссылки на внешние файлы проекта с JavaScript-кодом или непосредственно фрагменты этого кода.

При построении веб-страницы происходит обработка HTML-разметки документа, в процессе которой выполняется токенизация и построение дерева. Каждый токен состоит из открывающего и закрывающего тега и имеет некоторый набор атрибутов. Парсер считывает все входящие токены и преобразует их в документ, после чего строит для него DOM-дерево.

Параллельно с построением объектной модели документа происходит обработка CSS и построение дерева CSSOM (CSS Object Model). Деревья DOM и CSSOM являются независимыми. Сначала браузер преобразует все файлы CSS в карту стилей, после чего последовательно считывает каждое CSS-правило и создает дерево узлов на основе CSS-селекторов. Таким образом, построение CSSOM-дерева аналогично формированию DOM-дерева.

На основе построенных DOM и CSSOM деревьев браузер применяет CSS-стили к элементам HTML-разметки. Сначала применяются стили пользовательского агента (User Agent Stylesheet), определяющие внешний вид элементов по умолчанию. После этого браузер применяет наиболее общие CSS-правила для каждого узла, постепенно переходя к более специфичным правилам. В этом заключается механизм каскадирования CSS-стилей.

Помимо DOM и CSSOM деревьев браузер строит дерево доступности. Оно используется голосовыми помощниками и считывателями экрана для улучшения восприятия и интерпретирования содержимого веб-страницы. Дерево доступности представляет собой объектную модель доступности (Accessibility Object Model, AOM) и является семантической версией DOM. Обновление AOM происходит одновременно с обновлением DOM, но AOM-дерево, в отличие от DOM-дерева, не может быть изменено с помощью вспомогательных технологий.

После построения DOM-дерева происходит загрузка файлов JavaScript, по окончании которой выполняется интерпретация, компиляция, обработка и выполнение кода. Все скрипты преобразовываются в абстрактное синтаксическое дерево (Abstract Syntax Tree, AST). Этап компиляции заключается в передаче AST в интерпретатор для преобразования в байт-код, который будет исполняться в основном потоке.

Построение деревьев необходимо для внутреннего представления документа в браузере. Рендеринг страницы для отображения на экране пользователя включает в себя несколько этапов: стилизацию, компоновку, отрисовку и иногда композицию.

На этапе стилизации DOM и CSSOM деревья комбинируются в дерево рендеринга, на основе которого будет вычисляться положение всех видимых элементов страницы. Построение этого дерева осуществляется при прохождении DOM-дерева от корневого элемента с одновременной проверкой видимости каждого узла. Тег `<head>`, его дочерние элементы, а также элементы, скрытые с помощью стилей или скриптов, не включаются в дерево рендеринга.

Поскольку построенное дерево содержит все видимые элементы, их содержимое и стили, но не содержит информацию о геометрии узлов, то после этапа стилизации выполняется этап компоновки. В ходе него происходит вычисление размеров и положения всех узлов дерева рендеринга. Для этого браузер выполняет обход дерева. С учетом размера видимой области выполняется вычисление геометрии для всех элементов, начиная с тега `<body>` и переходя к его потомкам. При загрузке элемента, изменяющего вычисленные значения, запускается процесс повторного вычисления размеров и позиции узлов.

После этапа компоновки запускается этап отрисовки каждого отдельного узла на экране пользователя. В процессе отрисовки браузер конвертирует все узлы дерева в пиксели на экране с учетом информации, полученной на предыдущих этапах. Отрисовка элементов занимает весь основной поток для обеспечения плавной анимации и прокрутки страницы. Элементы дерева могут быть разбиты на слои для ускорения рендеринга путем переноса отрисовки некоторых слоев на GPU вместо CPU.

В случае, когда фрагменты страницы отрисованы на различных слоях, запускается этап композиции. Этот этап используется для определения наложения слоев и позволяет гарантировать, что все слои будут отрисованы на экране в нужном порядке, а их содержимое будет отображаться правильно.

При использовании Blazor процесс построения страниц происходит немного иначе. Внутри `<body>` резервируется специальный тег или компонента, в которой отображается сообщение о процессе загрузки страницы. Сначала происходит построение и отрисовка веб-страницы с указанным сообщением по описанному ранее алгоритму. После этого браузер переходит в режим ожидания до завершения построения настоящей страницы средствами Blazor.

Построение контента страницы в Blazor происходит в несколько этапов. Сначала выполняется инициализация необходимых параметров приложения и задание настроек отображения. После этого создаются и инициализируются необходимые менеджеры и сервисы для работы с компонентами и содержащимися в них данными. Далее запускается динамическое построение дерева документа, в процессе которого задаются значения всех параметров, используемых внутри компонент, после чего все компоненты превращаются в соответствующую им HTML-разметку. Полученный текст HTML загружается внутрь зарезервированного тега, заменяя находящееся в нем сообщение о загрузке. После этого браузер выполняет перестроение страницы с учетом обновленного содержимого.

2.2. СОЗДАНИЕ КОМПОНЕНТОВ

Приложения, созданные с использованием Blazor, основаны на использовании компонентов. Компонентом может являться любой из элементов пользовательского интерфейса, например, кнопка, текстовое поле, форма, диалоговое окно или страница целиком.

Компоненты Blazor позволяют эффективно решать большинство задач, возникающих при разработке интерфейсов. Например, с их помощью можно обрабатывать пользовательские действия и выполнять валидацию входных данных. Компоненты делают логику визуализации интерфейса более гибкой и универсальной, а также помогают избежать дублирования кода путем их многократного использования в различных частях интерфейса. Кроме того, Blazor-компоненты могут распространяться в виде NuGet-пакетов или библиотек классов.

Компоненты Blazor представляют собой классы на языке C#, которые создаются в виде страниц разметки в файлах с расширением `razor`. Файл с разметкой содержит в себе теги HTML, формирующие элементы пользовательского интерфейса, и код на языке C#, реализующий их функциональность. Переключение между HTML и C# осуществляется с помощью символа `@`. Указание такого префикса позволяет вставлять произвольные блоки кода внутрь разметки и использовать различные операторы языка C#, такие как условные выражения, циклы и операторы обработки исключений. При сборке страницы выражения на C# вычисляются и отображаются в HTML, после чего компоненты преобразуются в обычную HTML-разметку, формируя DOM-дерево.

Работа с компонентами основана на использовании двух основных функций Razor: директив и их атрибутов — зарезервированных ключевых слов, начинающихся с символа `@`. Директивы и атрибуты позволяют получить доступ к расширенному списку дополнительных функций компонента. Представим описание директив и атрибутов, использованных в данной работе, в виде таблиц 1 и 2.

Таблица 1 – Директивы Razor

Название	Описание
<code>@code</code>	Добавляет внутрь компонента блок кода на языке C#. Допускается создание нескольких блоков кода. Добавление C# кода может быть также реализовано путем добавления дополнительного файла с именем компонента и расширением <code>razor.cs</code> . В таком случае класс внутри файла <code>razor.cs</code> должен быть разделяемым и содержать ключевое слово <code>partial</code> .
<code>@inherits</code>	Указывает базовый класс, от которого наследуется класс компонента.
<code>@page</code>	Указывает URL адрес, по которому будет отображаться страница.
<code>@inject</code>	Внедряет внутрь компонента службу, менеджер или сервис. Допускается внедрение с помощью ключевого слова <code>[Inject]</code> .
<code>@using</code>	Добавляет директиву <code>using</code> языка C#. Список используемых директив <code>using</code> может указываться в файлах <code>razor</code> и <code>razor.cs</code> для каждого компонента отдельно или для всего проекта в файле <code>_Imports.razor</code> .

Таблица 2 – Атрибуты директив Razor

Название	Описание
<code>@onchange</code>	Указывает функцию для обработки события изменения элемента интерфейса, содержащегося внутри компонента.
<code>@onclick</code>	Указывает функцию для обработки события нажатия клавиши мыши внутри элемента интерфейса, содержащегося внутри компонента.
<code>@bind-value</code>	Привязывает данные к компоненту.
<code>@ref</code>	Указывает ссылку на экземпляр компонента для возможности выполнения команды для этого экземпляра. При отрисовке компонента поле будет заполнено его экземпляром.
<code>@onclick:preventDefault</code>	Запрещает выполнение действия, предусмотренного для элемента интерфейса по умолчанию.
<code>@onclick:stopPropagation</code>	Останавливает распространение события нажатия мыши вверх по DOM-дереву от текущего элемента интерфейса до корневого элемента страницы.

Именованное компонентов производится с использованием соглашения PascalCase, согласно которому слова не разделяются пробелами и знаками препинания, и каждое слово начинается с заглавной буквы. Если компонент находится внутри папки, отличной от корневой, то перед его именем указывается пространство имен в виде полного пути с разделением названий папок с помощью точек (например, `Shared.Components.Inputs.TextInput`). Чтобы использовать имя без указания пространства имен, следует добавить директиву `@using` с названием пути до компонента в файл `_Imports.razor`.

Каждый компонент может иметь произвольный список параметров, используемых для передачи данных. Объявление параметра осуществляется с помощью ключевого слова `[Parameter]` внутри блока `@code` или файла с расширением `razor.cs`. Параметры являются публичными свойствами класса компонента, могут иметь любой тип и название, а также значение по умолчанию.

2.3. ВЗАИМОДЕЙСТВИЕ С JAVASCRIPT ИЗ BLAZOR

Приложения Blazor позволяют организовывать взаимодействие с JavaScript посредством вызова его функций из методов C#. Такое взаимодействие называется JavaScript Interoperability (JS Interop). Код JavaScript или ссылка на файл со скриптами размещается внутри тега `<script>`. При этом тег `<script>` не может размещаться внутри файла компонента Razor, поскольку он не может быть изменен динамически. Программный код пользовательских скриптов может быть загружен в приложение Blazor несколькими способами.

Загрузка скриптов может осуществляться внутри тега `<head>` главной html-страницы. Такой подход не рекомендуется к использованию из-за особенностей работы парсера веб-страниц в браузере. Если парсер в процессе обработки разметки находит ресурсы, не препятствующие разбору документа, браузер отправляет запрос для загрузки соответствующих ресурсов и продолжает работу. Например, изображения и ссылки на CSS-файлы не блокируют работу парсера, в то время как наличие тега `<script>` приостанавливает обработку HTML-разметки до завершения загрузки указанного скрипта.

Более предпочтительным считается размещение тегов `<script>` в конце тега `<body>`, а не внутри тега `<head>`, во избежание блокировки работы парсера и, как следствие, остановки загрузки содержимого веб-страницы.

Вставка JavaScript-кода непосредственно внутрь тега `<script>` является не самым эффективным вариантом. Рекомендуется располагать весь JavaScript-код в отдельном файле с расширением `js`, а внутрь атрибута `src` тега `<script>` размещать ссылку на соответствующий файл.

Последним способом загрузки является внедрение скриптов после запуска Blazor. Для этого следует добавить атрибут `autostart="false"` к тегу `<script>`, загружающему скрипт Blazor. Далее в конце `<body>` размещается скрипт, который ссылается на пользовательский файл сценариев после запуска приложения Blazor посредством вызова `Blazor.start().then(...)`.

Вызов функций JavaScript из методов C# осуществляется с помощью добавления экземпляра `IJSRuntime` внутрь класса компонента с помощью ключевого слова `[Inject]` или директивы `@inject`. Для вызова функции JavaScript из C# необходимо асинхронно вызвать один из методов `InvokeVoidAsync` или `InvokeAsync` с указанием необходимых параметров.

Метод `InvokeVoidAsync` следует использовать в тех случаях, когда в методах C# не требуется чтение возвращаемого значения функции JavaScript. При этом функция JavaScript не должна возвращать значение. Пример вызова функции JavaScript из компонента Razor представлен на рисунке 1.

```
@inject IJSRuntime JS

@code
{
    async void Load() => await JS.InvokeVoidAsync("Close", "load");
}
```

Рисунок 1 – Фрагмент кода для вызова функции JavaScript из компонента Razor

Метод `InvokeAsync` применяется в случаях, когда в методах C# необходимо получение возвращаемого значения функции JavaScript. Пример вызова функции JavaScript из класса C# представлен на рисунке 2.

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Components;
using Microsoft.JSInterop;

namespace BlazorShared.Components.Blocks
{
    public partial class Popup
    {
        [Inject] public IJSRuntime JS { get; set; }

        public async ValueTask<double> CalculatePosition(string id)
        {
            return await JS.InvokeAsync<double>("CalculatePosition", id);
        }
    }
}
```

Рисунок 2 – Фрагмент кода для вызова функции JavaScript из класса C#

При взаимодействии с JavaScript из приложения Blazor хорошей практикой считается изоляция пользовательских скриптов в модулях JavaScript. Использование изоляции позволяет очистить глобальное пространство имен приложения путем импортирования JavaScript-кода. Рассмотрим пример изоляции пользовательского скрипта для открытия окна подтверждения очистки решения. Фрагменты кода экспортируемой JavaScript-функции и импортирующего ее компонента Razor представлены на рисунках 3 и 4 соответственно.

```
export function Confirm(question) {  
    return confirm(question);  
}
```

Рисунок 3 – Код экспортируемой функции в файле модуля `confirm.js`

```
@inject IJSRuntime JSInterop  
  
<Popup Id="clear-results" Title="Clear results">  
    @if (ConfirmResult) {  
        <Label Text="Solution was successfully cleared!" />  
    } else {  
        <Label Text="The problem has already been calculated." />  
        <TextButton Text="Clear" OnClickkEvent=@(() => ShowConfirm()) />  
    }  
    <TextButton Text="Cancel" OnClickkEvent=@(() => Cancel()) />  
</Popup>  
  
@code  
{  
    IJSObjectReference JSModule;  
    bool ConfirmResult = false;  
  
    protected override async Task OnAfterRenderAsync(bool firstRender)  
    {  
        if (firstRender)  
            JSModule = await JSInterop.InvokeAsync<IJSObjectReference>(  
                "import", "../js/confirm.js");  
    }  
  
    async Task ShowConfirm() =>  
        ConfirmResult = await JSModule.InvokeAsync<bool>(  
            "Confirm", "Are you sure to clear the results?");  
  
    async void Cancel() =>  
        await JSInterop.InvokeVoidAsync("ClosePopup", "clear-results");  
}
```

Рисунок 4 – Компонент всплывающего окна, вызывающий модуль JavaScript

Файл `confirm.js` является модулем JavaScript и экспортирует функцию `ShowConfirm`. Эта функция отображает в браузере пользователя модальное окно с указанным вопросом и кнопками подтверждения и отмены. Пользователь не сможет совершать никаких действий в браузере, пока это окно не будет закрыто. Результатом выполнения функции является `true`, если была нажата кнопка подтверждения, и `false` в противном случае.

Если задача была посчитана, то при нажатии на кнопку очистки результатов появляется всплывающее окно, реализованное в виде компонента `Popup`. Это окно содержит предупреждение о существовании решения, а также кнопки `Clear` и `Cancel` для его очистки и закрытия окна соответственно.

При загрузке компонента окна `Popup` модуль JavaScript импортируется в приложение с помощью вызова асинхронного метода `InvokeAsync` экземпляра `IJSRuntime`, поскольку для динамического импортирования модуля требуется выполнение сетевого запроса. Для вызова экспортированных функций из модуля JavaScript используется тип `IJSObjectReference`, представляющий собой ссылку на объект JavaScript из метода C#. Идентификатор `import` используется специально для импорта JavaScript-модулей.

При нажатии на кнопку `Clear` вызывается C# метод `ShowConfirm`, вызывающий JavaScript-функцию `Confirm` для вызова модального окна из импортированного модуля. Результат выполнения функции сохраняется, поэтому в случае подтверждения в модальном окне в компонент `Popup` выводится сообщение об успешном выполнении очистки решения.

Взаимодействие с JavaScript из Blazor может быть невозможно во время предварительной подготовки веб-приложения. Чтобы задержать вызов функций JavaScript до момента их гарантированного выполнения, следует переопределить в коде компонента событие `OnAfterRender` или `OnAfterRenderAsync` и вызывать необходимые пользовательские скрипты внутри одного из них. Вызов этих событий осуществляется только после полной загрузки и визуализации приложения Blazor.

3. КАТАЛОГ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Целью данной работы является перевод интерфейса приложения, созданного с использованием WPF (Windows Presentation Foundation), в интерфейс веб-приложения на основе технологии Blazor [11]. В качестве основы для создания веб-версии View была выбрана двумерная часть программного комплекса Telma. Реализуем в приложении Blazor аналоги основных элементов управления Telma.

Разметка интерфейса WPF приложения создается с помощью языка разметки XAML (Extensible Application Markup Language) [12]. Для написания разметки веб-интерфейса приложения будем использовать HTML. Стилизация элементов интерфейса будет выполняться с помощью CSS.

По умолчанию в приложениях Blazor подключен CSS-фреймворк Bootstrap, содержащий набор HTML и CSS шаблонов для оформления типографики и некоторых элементов веб-интерфейса. Создание интерфейсов с помощью Bootstrap основано на использовании готовых CSS-классов с заранее определенным набором свойств, описывающих внешний вид элементов управления. Фреймворк Bootstrap используется в основном при создании простых веб-интерфейсов, поэтому он плохо подходит для разработки сложного с точки зрения интерактивности и функциональности интерфейса Telma. Кроме того, Telma обладает собственным уникальным дизайном, а с помощью шаблонов Bootstrap можно создавать веб-интерфейсы лишь в заданной фреймворком стилистике.

Поскольку стандартный для приложений Blazor фреймворк Bootstrap не подходит для реализации View на основе Telma, будем создавать все элементы пользовательского интерфейса самостоятельно, используя HTML и CSS в чистом виде. При разработке будем использовать компонентный подход.

Перейдем к описанию основных правил построения страниц веб-приложения и каталогизации реализованных компонентов с указанием списка возможных атрибутов.

3.1. ОБЩИЕ ПРАВИЛА СОЗДАНИЯ КОМПОНЕНТОВ

Существует несколько ситуаций, при которых реализованный набор компонентов является недостаточным для создания полноценного пользовательского интерфейса. В таких случаях требуется создание новых компонентов в дополнение к уже существующим. Рассмотрим основные ситуации, при которых добавление компонентов будет оправдано:

1. Новый компонент должен иметь сложную структуру и объединять некоторое количество других компонентов для выполнения заданного набора функций. К списку таких компонентов относятся всплывающие окна для интерактивного взаимодействия с пользователем, окна для ввода параметров и панели в верхней и боковой части экрана.

2. Какой-либо фрагмент разметки используется в нескольких частях приложения. Создание нового компонента поможет избежать многократного дублирования кода и сократить его размер, а также позволит при необходимости вносить правки в единственном месте приложения.

3. Некоторая часть разметки должна иметь ViewModel, отличающуюся от ViewModel содержащего ее компонента. В таком случае указанный фрагмент разметки следует сделать новым компонентом, вложенным по отношению к содержащему его компоненту. Класс вложенного компонента будет наследоваться от другого базового класса, совпадающего с необходимой для этого компонента ViewModel. К этому разделу можно также отнести ситуацию, когда вложенный компонент должен обновляться при наступлении события движения мыши, но родительский компонент не должен реагировать на это событие во избежание лишних обновлений интерфейса.

4. Требуется наличие текстового поля для обработки нового типа данных.

5. Необходим элемент управления одного из существующих типов, но с другим поведением.

6. Требуется новый элемент управления пользовательского интерфейса, отсутствующий в существующем наборе компонентов.

Опишем общие правила для привязки данных и задания атрибутов компонентов. Статические значения атрибутов описываются в кавычках и добавляются после имени атрибута и символа присваивания, как и в разметке HTML. Указание переменных и команд `ViewModel` в качестве значений атрибутов осуществляется в соответствии с синтаксическими правилами создания явных и неявных выражений Razor.

Неявные выражения имеют простую структуру и представляют собой обращение к свойству или методу класса. Такие выражения состоят из префикса `@` и последующего C# кода. При описании неявных выражений не допускается использование пробелов кроме случаев асинхронного вызова методов с помощью ключевого слова `await`. Неявные выражения также не должны содержать шаблоны C#, поскольку символы внутри угловых скобок интерпретируются Blazor как HTML-тег.

Для использования лямбда-функций, тернарных операторов, условных конструкций и других сложных выражений в качестве значения атрибута компонента Razor следует руководствоваться правилами создания явных выражений. Явные выражения состоят из префикса `@` и выражения на языке C#, заключенного в круглые скобки. Использование явных выражений позволяет объединять результат выполнения функции с дополнительным текстом. Кроме того, при описании явных выражений допускается использование пробелов и универсальных шаблонов C#.

Атрибуты компонентов имеют различные типы значений. Атрибуты с типом значения `RenderFragment` используются для задания дочернего содержимого компонента, которое может включать в себя любые компоненты и теги HTML или быть пустым. Такие атрибуты добавляются между открывающим и закрывающим тегами компонента. Дочернее содержимое атрибута добавляется между его открывающим и закрывающим тегами. Расположение атрибута с типом `RenderFragment` в разметке компонента Razor совпадает с местом отрисовки его дочернего содержимого в окончательной разметке HTML.

Привязка данных `ViewModel` к элементам управления интерфейса осуществляется посредством задания атрибутов для соответствующих компонентов и происходит в несколько этапов:

1. В начале файла компонента указывается директива `@inherits` со значением `ReactiveComponentBase<T>`, где `T` — название класса, реализующего `ViewModel` для соответствующего компонента `Razor`.

2. Необходимый атрибут компонента связывается с соответствующей переменной, командой или параметром состояния `ViewModel`. Переменные привязываются через атрибут `Source`, команды — через атрибут `Command`, а параметры состояния — через атрибуты `IsPushed` или `IsVisible`. Список возможных данных для привязки становится доступен с помощью вызова `ViewModel`. При использовании вложенных компонентов `ViewModel` для них должна передаваться в качестве параметра из родительского компонента.

3. В блоке `@code` для текстовых полей, выпадающих списков и чекбоксов создаются функции для обработки события изменения компонента, присваивающие новые значения нужных типов привязанным переменным. Созданные функции связываются с соответствующими компонентами через атрибут `OnChange`. Примеры привязки данных к различным компонентам приведены на рисунке 5.

```
<IntInput Source=@ViewModel.ProblemItem.AddBasisOrder
          OnChange=@OnAddBasisOrderChange />

<ToggleSectionButton Command=@ViewModel.OnTitleSetEditMode
                     IsPushed=@(ViewModel.OnTitleSetEditMode.IsChecked == true) />

<Checkbox Source=@ViewModel.ActiveTitleView.IsValueXDraw
          OnChange=@OnXComponentChange
          IsVisible=@ViewModel.ActiveTitleView.IsVectorValue />

@code {
    void OnAddBasisOrderChange(int value) =>
        ViewModel.ProblemItem.AddBasisOrder = value;

    void OnXComponentChange(bool value) =>
        ViewModel.ActiveTitleView.IsValueXDraw = value;
}
```

Рисунок 5 – Привязка данных из `ViewModel` к компонентам `Razor`

Для дальнейшего изложения введем некоторые комментарии к документированию созданных компонентов. Описание атрибутов компонентов представлено в табличной форме. В названии каждой таблицы указано имя компонента, содержащего описываемые атрибуты. При наличии аналога элемента в Telma его название также указывается в названии таблицы в круглых скобках рядом с именем компонента. При наличии аналога атрибута в Telma его название указывается в круглых скобках рядом с названием атрибута компонента Razor.

Опишем в каталоге все созданные компоненты и их атрибуты с примерами, приведем графическое представление компонентов при различных состояниях, а также при необходимости укажем причины самостоятельного создания элементов управления вместо использования стандартных элементов HTML. Все компоненты в каталоге разбиты по разделам, исходя из их назначения и типа обрабатываемых значений.

Для каждого атрибута компонента опишем его название, аналог в Telma, тип принимаемого значения, список возможных значений с расшифровкой и значение по умолчанию. Для некоторых компонентов существуют обязательные атрибуты, без которых элементы управления интерфейса будут работать некорректно. Обязательные атрибуты компонентов будут выделены в таблицах с помощью подчеркивания.

Некоторые компоненты Razor могут представлять единый элемент управления, но при этом определять несколько его вариаций для использования в различных частях страниц приложения. Выбор одного из возможных представлений элемента управления осуществляется с помощью указания соответствующего значения в атрибуте типа компонента. Наличие нескольких вариантов одного элемента может быть обусловлено их различным поведением, визуальным оформлением или структурой. Различные типы компонента имеют различные наборы атрибутов. Для каждого типа компонента будем описывать набор его атрибутов в виде отдельной таблицы.

Примеры текстов программ для различных типов элементов управления представлены в приложениях А–К.

3.2. ЭЛЕМЕНТЫ УПРАВЛЕНИЯ ДЛЯ ВВОДА И ВЫБОРА ДАННЫХ

3.2.1. ПОЛЯ ДЛЯ ТЕКСТА И ЧИСЛОВЫХ ЗНАЧЕНИЙ

Для работы с текстом и числовыми значениями существует несколько видов элементов управления:

1. `BaseTableDefinedProperty` — таблица для вывода значений коэффициента задачи (таблица 3).
2. `TextInput` — поле для короткого текста (таблицы 4 и 5).
3. `Textarea` — поле для многострочного текста (таблица 6).
4. `Quantity` — поле для целого числа с кнопками изменения (таблица 7).
5. `MHSplineDefinedProperty` — поле для названия загруженной кривой, используемой при решении нелинейной задачи (таблица 8).
6. `DoubleInput` — поле для вещественного значения (таблицы 9 и 10).
7. `IntInput` — поле для целочисленного значения (таблицы 11 и 12).
8. `ParameterPoint2DInput` — поле для 2D-вектора (таблицы 13 и 14).
9. `ParameterPoint3DInput` — поле для 3D-вектора (таблицы 15 и 16).
10. `TextProperty` — поле для чисел в текстовом формате (таблица 17).
11. `EditableListItem` — поле для короткого текста с набором кнопок для модификации, добавления и удаления элементов управления (таблица 18).
12. `PointInput` — набор полей для векторов в декартовой и полярной системах координат и для вектора со значением локального нуля окна (таблица 19).

Таблица 3 – `BaseTableDefinedProperty` (`DataTemplate` с `DataGrid`)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
<code>IsVisible</code> (<code>Visibility</code>)	<code>bool</code>	Видимость поля	<code>true</code> <code>false</code>	<code>true</code>
<pre><BaseTableDefinedProperty ViewModel=@((BaseTableDefinedProperty) (ViewModel.Property)) /></pre>				

Таблица 4 – TextInput (TelmaTextBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text	string	Подпись поля	Любой текст	""
Source (Text)	string	Значение в поле	Любой текст	""
OnChange	Action<string>	Обработчик события изменения	Callback-функция	null
SizeType	Enum InputSize	Размер поля	Short Middle Long Stretch	Stretch
IsReadonly (IsReadOnly)	bool	Доступность только для чтения	true false	false
IsVisible (Visibility)	bool	Видимость поля	true false	true
<pre><TextInput Text="File name:" Source=@ViewModel.MeshItem.File OnChange=@OnFileNameChange SizeType=@InputSize.Long IsVisible="false" /></pre>				

Таблица 5 – TextInput с InputType = Table (TelmaTextBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Source (Text)	string	Значение в поле	Любой текст	""
OnChange	Action<string>	Обработчик события изменения	Callback-функция	null
IsReadonly (IsReadOnly)	bool	Доступность только для чтения	true false	false
IsVisible (Visibility)	bool	Видимость поля	true false	true
<pre><TextInput InputType=@InputType.Table Source=@ViewModel.Name OnChange=@OnVariableNameChange IsVisible="false" /></pre>				

Таблица 6 – Textarea (TextBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Source (Text)	string	Текст в поле	Любой текст	""
Rows	int	Число строк	Целое число	10
Columns	int	Число столбцов	Целое число	1
OnChange	Action<string>	Обработчик события изменения	Callback-функция	null
IsReadOnly (IsReadOnly)	bool	Доступность только для чтения	true false	false
IsVisible (Visibility)	bool	Видимость поля	true false	true
<pre><Textarea Rows="3" Source=@ViewModel.TextEditor OnChange=@OnTextEditorChange /></pre>				

Таблица 7 – Quantity (TextBox + ScrollBar)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Source (Text + Value)	int	Значение в поле	Целое число	0
Min (Minimum)	int	Минимальное значение	Целое число	-10000
Max (Maximum)	int	Максимальное значение	Целое число	10000
Step (SmallChange)	int	Шаг изменения значения	Целое число	1
OnChange	Action<int>	Обработчик события изменения	Callback-функция	null
IsVisible (Visibility)	bool	Видимость поля	true false	true
<pre><Quantity IsVisible=@LogScaleX Source=@IXLogOrder OnChange=@OnIXLogOrderChange Min="-100" Max="100" Step="2" /></pre>				

Таблица 8 – MHSplineDefinedProperty (DataTemplate с Label + TextBlock)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
IsVisible (Visibility)	bool	Видимость поля	true false	true
<code><MHSplineDefinedProperty ViewModel=@ ((MHSplineDefinedProperty) (ViewModel.Property)) /></code>				

Таблица 9 – DoubleInput (TelmaScalarBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text	string	Подпись поля	Любой текст	""
Source (DblValue)	ParameterFloat	Значение в поле	Вещественное число	0.0
Min (Min)	ParameterFloat	Минимальное значение	Вещественное число	-10000.0
Max (Max)	ParameterFloat	Максимальное значение	Вещественное число	10000.0
OnChange	Action <ParameterFloat>	Обработчик события изменения	Callback-функция	null
SizeType	Enum InputSize	Размер поля	Short Middle Long Stretch	Stretch
IsReadOnly (IsReadOnly)	bool	Доступность только для чтения	true false	false
IsVisible (Visibility)	bool	Видимость поля	true false	true
<code><DoubleInput Text="Residual:" SizeType=@InputSize.Middle Source=@ViewModel.Settings.SlaeParameters.SLAEEps OnChange=@OnLinearResidualChange Min="0.0" Max="1.0" /></code>				

Таблица 10 – DoubleInput с InputType = Table (TelmaScalarBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Source (DblValue)	ParameterFloat	Значение в поле	Вещественное число	0.0
Min (Min)	ParameterFloat	Мин. значение	Вещественное число	-10000.0
Max (Max)	ParameterFloat	Макс. значение	Вещественное число	10000.0
OnChange	Action <ParameterFloat>	Обработчик события изменения	Callback-функция	null
IsReadOnly (IsReadOnly)	bool	Доступность только для чтения	true false	false
IsVisible (Visibility)	bool	Видимость поля	true false	true
<code><DoubleInput InputType=@InputType.Table Source=@ViewModel.BegT OnChange=@OnBegTChange /></code>				

Таблица 11 – IntInput с InputType = Table (TelmaIntBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Source (IntValue)	int	Значение в поле	Целое число	0
Min (Min)	int	Минимальное значение	Целое число	-10000
Max (Max)	int	Максимальное значение	Целое число	10000
OnChange	Action<int>	Обработчик события изменения	Callback-функция	null
IsReadOnly (IsReadOnly)	bool	Доступность только для чтения	true false	false
IsVisible (Visibility)	bool	Видимость поля	true false	true
<code><IntInput InputType=@InputType.Table OnChange=@OnTimeInternalChange Source=@ViewModel.MeshTimeElement.Internal /></code>				

Таблица 12 – IntInput (TelmaIntBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text	string	Подпись поля	Любой текст	""
Source (IntValue)	int	Значение в поле	Целое число	0
Min (Min)	int	Мин. значение	Целое число	-10000
Max (Max)	int	Макс. значение	Целое число	10000
OnChange	Action<int>	Обработчик события изменения	Callback-функция	null
SizeType	Enum InputSize	Размер поля	Short Middle Long Stretch	Stretch
IsReadOnly (IsReadOnly)	bool	Доступность только для чтения	true false	false
IsVisible (Visibility)	bool	Видимость поля	true false	true
<pre><IntInput Text="Internal =" Source=@ViewModel.Internal Min="0" Max="5" SizeType=@InputSize.Short OnChange=@OnInternalChange /></pre>				

Таблица 13 – ParameterPoint2DInput с InputType = Table (TelmaPoint2DBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Source (PointValue)	ParameterPoint2D	Значение в поле	Вектор 2D	(0, 0)
OnChange	Action <ParameterPoint2D>	Обработчик события изменения	Callback-функция	null
IsReadOnly (IsReadOnly)	bool	Доступность только для чтения	true false	false
IsVisible (Visibility)	bool	Видимость поля	true false	true
<pre><ParameterPoint2DInput InputType=@InputType.Table Source=@ViewModel.ArcCenter IsReadOnly="true" /></pre>				

Таблица 14 – ParameterPoint2DInput (TelmaPoint2DBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text	string	Подпись поля	Любой текст	" "
Source (PointValue)	ParameterPoint2D	Значение в поле	Вектор 2D	(0, 0)
OnChange	Action <ParameterPoint2D>	Обработчик события изменения	Callback-функция	null
SizeType	Enum InputSize	Размер поля	Short Middle Long Stretch	Stretch
IsReadonly (IsReadOnly)	bool	Доступность только для чтения	true false	false
IsVisible (Visibility)	bool	Видимость поля	true false	true
<ParameterPoint2DInput Text="Arc center:" Source=@ViewModel.ArcCenter SizeType=@InputSize.Long IsReadonly="true" />				

Таблица 15 – ParameterPoint3DInput с InputType = Table (TelmaPoint3DBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Source (PointValue)	ParameterPoint3D	Значение в поле	Вектор 3D	(0, 0, 0)
OnChange	Action <ParameterPoint3D>	Обработчик события изменения	Callback-функция	null
IsReadonly (IsReadOnly)	bool	Доступность только для чтения	true false	false
IsVisible (Visibility)	bool	Видимость поля	true false	true
<ParameterPoint3DInput InputType=@InputType.Table Source=@ViewModel.Center IsReadonly="true" />				

Таблица 16 – ParameterPoint3DInput (TelmaPoint3DBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text	string	Подпись поля	Любой текст	""
Source (PointValue)	ParameterPoint3D	Значение в поле	Вектор 3D	(0, 0, 0)
OnChange	Action <ParameterPoint3D>	Обработчик события изменения	Callback-функция	null
SizeType	Enum InputSize	Размер поля	Short Middle Long Stretch	Stretch
IsReadonly (IsReadOnly)	bool	Доступность только для чтения	true false	false
IsVisible (Visibility)	bool	Видимость поля	true false	true
<code><ParameterPoint3DInput Text="Center:" Source=@ViewModel.Center SizeType=@InputSize.Long IsReadonly="true" /></code>				

Таблица 17 – TextProperty (DataTemplate с TelmaTextBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text	string	Подпись поля	Любой текст	""
SizeType	Enum InputSize	Размер поля	Short Middle Long Stretch	Stretch
IsReadonly (IsReadOnly)	bool	Доступность только для чтения	true false	false
IsVisible (Visibility)	bool	Видимость поля	true false	true
<code><TextProperty Text=@ViewModel.Name SizeType=@InputSize.Middle ViewModel=@ ((BaseTextDefinedProperty) (ViewModel.Property)) /></code>				

Таблица 18 – EditableListItem (TelmaEditableListItem)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text	string	Подпись поля	Любой текст	"Name ="
Value	string	Значение в поле	Любой текст	" "
AddElementEvent (AddElementEvent)	Action <ListItemData>	Обработчик события добавления	Callback-функция	null
EditElementEvent (EditElementEvent)	Action <ListItemData>	Обработчик события изменения	Callback-функция	null
DeleteElementEvent (DeleteElementEvent)	Action <ListItemData>	Обработчик события удаления	Callback-функция	null
IsVisible (Visibility)	bool	Видимость поля и всех кнопок	true false	true
IsAddButtonVisible (IsAddButtonPresent)	bool	Видимость кнопки добавления	true false	true
IsEditButtonVisible (IsEditButtonPresent)	bool	Видимость кнопки редактирования	true false	true
IsDeleteButtonVisible (IsDeleteButtonPresent)	bool	Видимость кнопки удаления	true false	true
<EditableListItem Value=@ViewModel.Name AddElementEvent=@AddMeshEvent DeleteElementEvent=@DeleteMeshEvent EditElementEvent=@EditMeshEvent />				

Таблица 19 – PointInput (PointEditControl)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
OnChange	Action <ParameterPoint3D>	Обработчик события изменения	Callback-функция	null
IsReadOnly (IsReadOnly)	bool	Доступность только для чтения	true false	false
IsVisible (Visibility)	bool	Видимость поля	true false	true
<PointInput Source=@ViewModel.EndPoint.As3D() OnChange=@OnEndChange />				

Описанные элементы управления представлены на рисунке 6.

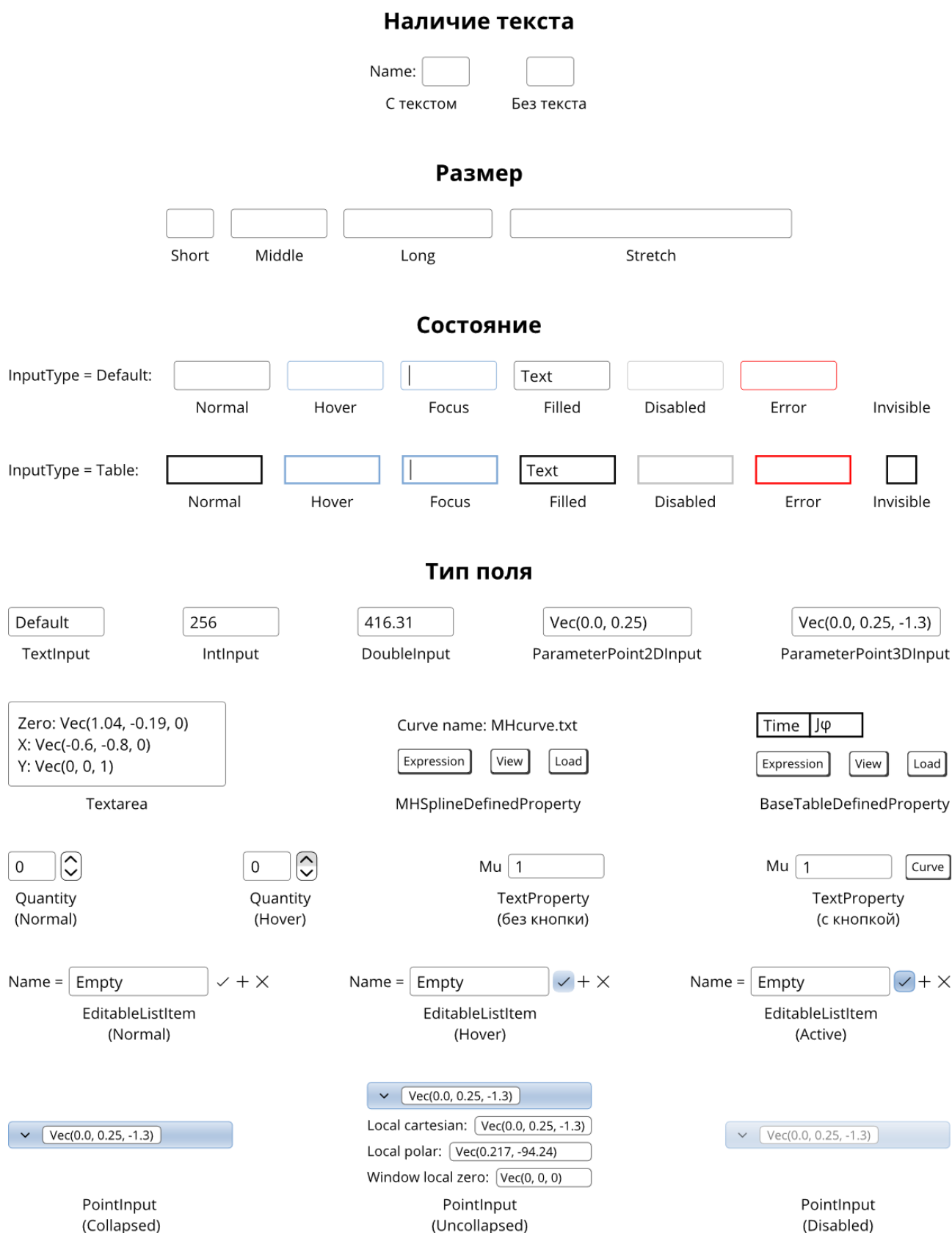


Рисунок 6 – Визуальное представление полей для текста и числовых значений

3.2.2. ВЫПАДАЮЩИЕ СПИСКИ ДЛЯ ВЫБОРА ЗНАЧЕНИЯ

Выпадающий список для выбора значений реализован в веб-интерфейсах по умолчанию. Для его создания используется HTML-тег `<select>`, внутри которого располагаются теги `<option>`, представляющие собой значения списка. Выбранное пользователем значение становится активным и выводится в элемент управления `<select>`, заменяя предыдущее значение. Стандартный выпадающий список обладает рядом серьезных недостатков, снижающих удобство работы с интерфейсом приложения и не позволяющих реализовать необходимый функционал MVVM View на основе Telma.

Ширина стандартного элемента `<select>` всегда равна максимальной ширине элемента списка и не подстраивается по ширине выбранного пользователем значения. Таким образом, если выбран элемент малой ширины, но при этом в списке имеется значение большой ширины, то элемент `<select>` будет иметь много пустого пространства, затрудняющего взаимодействие с интерфейсом.

Кроме того, инструментарий веб-технологий не позволяет изменять внешний вид тегов `<option>`, а также добавлять внутрь них другие элементы, отличные от текста. В программном комплексе Telma некоторые выпадающие списки состоят из значений, представленных изображениями.

С учетом описанных недостатков оптимальным вариантом является самостоятельное создание аналога выпадающего списка взамен реализованного в HTML по умолчанию. При создании аналога необходимо повторить поведение стандартного элемента `<select>`, а также дополнить и улучшить его функциональность. Для реализованного списка добавлены следующие возможности:

1. Открытие и закрытие списка с выбором значения.
2. Открытие списка вниз и вверх.
3. Открытие списка поверх всего имеющегося контента страницы.
4. Установка ширины закрытого списка в соответствии с шириной выбранного значения.
5. Выравнивание блока со значениями по левой и правой стороне.

Для работы с выпадающими списками реализовано несколько типов компонентов в соответствии с их назначением:

1. `Select` — выпадающий список с совпадающими для `View` и `ViewModel` текстовыми значениями (таблица 20).

2. `ImageSelect` — выпадающий список со значениями, представленными изображениями (таблица 21).

3. `DictionarySelect` — выпадающий список, сопоставляющий различные для `View` и `ViewModel` текстовые значения (таблица 22).

4. `ScalarValueSelector` — выпадающий список для вывода названий физических величин, для которых можно построить распределение поля. Для векторных величин также становится доступен выпадающий список с названиями компонент вектора.

Таблица 20 – `Select` (ComboBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
<code>Source</code> (<code>ItemsSource</code>)	<code>IEnumerable<string></code>	Список значений	Коллекция текстовых строк	<code>null</code>
<code>OnChange</code>	<code>Action<string></code>	Обработчик события изменения	Callback-функция	<code>null</code>
<code>Selected</code> (<code>SelectedValue</code>)	<code>string</code>	Выбранное значение	Любой текст	<code>""</code>
<code>Title</code> (<code>ToolTip</code>)	<code>string</code>	Всплывающая подсказка	Любой текст	<code>""</code>
<code>Orientation</code>	<code>Enum Orientation</code>	Направление открытия списка	<code>Bottom</code> <code>Top</code>	<code>Bottom</code>
<code>Side</code>	<code>Enum Side</code>	Выравнивание списка	<code>Left</code> <code>Right</code>	<code>Left</code>
<pre><Select Selected=@ViewModel.ConditionName Source=@BoundaryNames OnChange=@OnBoundaryChange Side=@Side.Right Orientation=@Orientation.Top /></pre>				

Таблица 21 – ImageSelect (ComboBox + Array, содержащий набор Image)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Source (ItemsSource)	Dictionary <int, (string, string)>	Список значений (номер, (пояснение, картинка))	(целое число, (текст, текст))	null
OnChange	Action<int>	Обработчик события изменения	Callback-функция	null
Selected (SelectedIndex)	int	Выбранное значение	Целое число	0
Title (ToolTip)	string	Всплывающая подсказка	Любой текст	" "
Orientation	Enum Orientation	Направление открытия списка	Bottom Top	Bottom
Side	Enum Side	Выравнивание списка	Left Right	Left
<pre><ImageSelect Source=@(new Dictionary<int, (string, string)>() { [0] = ("XY projection", "img/projection/Front.png") }) /></pre>				

Таблица 22 – DictionarySelect (ComboBox + [Array или набор ComboBoxItem])

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Source (ItemsSource)	Dictionary <string, string>	Список значений	Коллекция строк	null
OnChange	Action<string>	Обработчик события изменения	Callback-функция	null
Selected (SelectedValue)	string	Выбранное значение	Любой текст	" "
Title (ToolTip)	string	Всплывающая подсказка	Любой текст	" "
Orientation	Enum Orientation	Направление открытия списка	Bottom Top	Bottom
Side	Enum Side	Выравнивание списка	Left Right	Left
<pre><DictionarySelect Selected=@ViewModel.XSymmetry.ToString() Source=@(new Dictionary<string, string>() { ["0"] = "None" }) /></pre>				

Описанные типы выпадающих списков в сравнении со стандартным HTML элементом `<select>` представлены на рисунке 7.

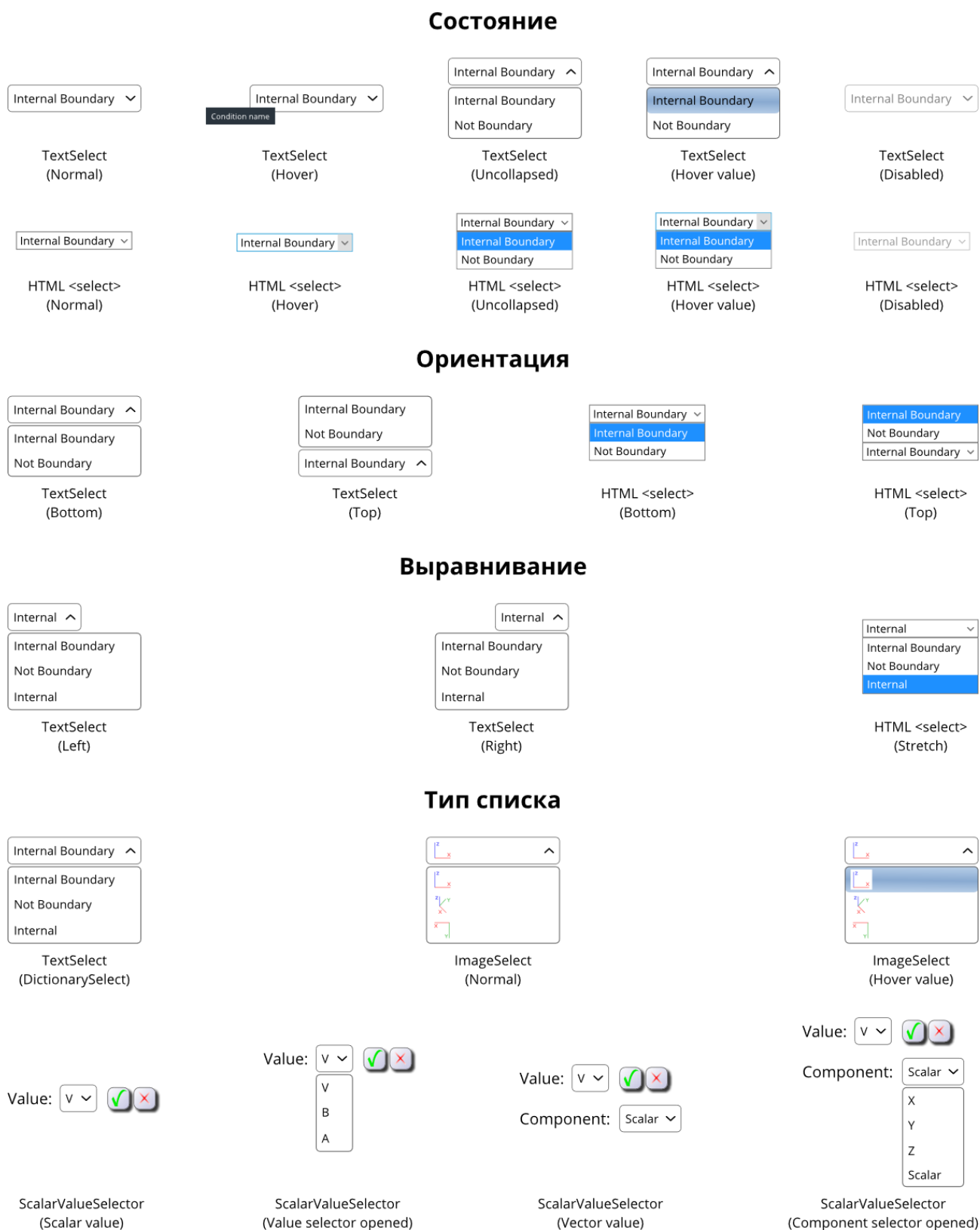


Рисунок 7 – Визуальное представление выпадающих списков

3.2.3. ЭЛЕМЕНТЫ УПРАВЛЕНИЯ ДЛЯ ВЫБОРА ЗНАЧЕНИЙ

Для выбора цветов, значений и состояния параметров реализовано несколько типов компонентов:

1. `ColorPicker` — элемент управления интерфейса для выбора цвета в одном из форматов RGB (Red-Green-Blue, цветовая модель для кодирования цвета с помощью трех основных цветов), HSL (Hue-Saturation-Lightness, цветовая модель для кодирования цвета с помощью задания тона, насыщенности и светлоты) или HEX (три пары шестнадцатеричных цифр, отвечающих за отдельный цвет). Помимо полей для задания числовых значений цвета элемент `ColorPicker` также включает в себя цветовую палитру и инструмент для выбора цвета с экрана. Элемент реализован на основе стандартного HTML элемента с тегом `<input>` и типом `color` (таблица 23).

2. `Checkbox` — элемент интерфейса для выбора активного или неактивного состояния параметра или множественного выбора значений из списка. Каждый элемент `Checkbox` должен иметь уникальное значение атрибута `Id`. Чекбокс с типом `CheckboxType = Input` используется в сочетании с элементом управления `PointInput` и имеет такой же внешний вид, поведение и набор атрибутов, как и обычный `Checkbox` с типом `CheckboxType = Default`. Элемент реализован на основе стандартного HTML элемента с тегом `<input>` и типом `checkbox` в сочетании с HTML элементом `<label>` (таблица 24).

3. `RadioButton` — элемент управления, позволяющий выбирать одно из значений заранее определенного набора. Взаимоисключающие радиокнопки из одного списка должны иметь одинаковое значение атрибута `Name`. Активация какой-либо радиокнопки из списка автоматически отменяет активное состояние другой радиокнопки в этом списке. По умолчанию ни одна из радиокнопок не является активной. Элемент реализован на основе стандартного HTML элемента с тегом `<input>` и типом `radio` в сочетании с HTML элементом `<label>` (таблица 25).

Таблица 23 – ColorPicker

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Source	Color	Выбранный цвет	Любой цвет	—
IsVisible	bool	Видимость элемента	true false	true
OnChange	Action<Color>	Обработчик события изменения	Callback-функция	null
<pre><ColorPicker Source=@mat.Color OnChange=@((color) => MaterialColorEdit(mat.DisplayName, color)) /></pre>				

Таблица 24 – Checkbox (CheckBox или RibbonCheckBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text (Content Label)	string	Текст в чекбоксе	Любой текст	""
Source (IsChecked)	bool	Состояние параметра	true false	false
ChildContent	RenderFragment	Вложенный контент	Компоненты или теги HTML	null
<u>Id</u>	string	Идентификатор чекбокса	Любой текст без пробелов	""
OnChange	Action<bool>	Обработчик события изменения	Callback-функция	null
CheckboxSize	Enum CheckboxSize	Размер чекбокса	Small Default	Default
IsEnabled (IsEnabled)	bool	Доступность для взаимодействия	true false	true
IsVisible (Visibility)	bool	Видимость чекбокса	true false	true
<pre><Checkbox Source=@ViewModel.ActiveTitleView.IsValueScalarDraw Id="value-scalar" OnChange=@OnScalarChange Text="Scalar" CheckboxSize=@CheckboxSize.Small IsVisible=@ViewModel.ActiveTitleView.IsVectorValue /></pre>				

Таблица 25 – RadioButton

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text	string	Текст внутри радиокнопки	Любой текст	""
Source	bool	Активность элемента	true false	false
ChildContent	RenderFragment	Вложенный контент	Компоненты или теги HTML	null
<u>Name</u>	string	Имя для связи с другими радиокнопками	Любой текст без пробелов	""
OnChange	Action<bool>	Обработчик события изменения	Callback-функция	null
RadioButtonSize	Enum RadioButtonSize	Размер радиокнопки	Small Default	Default
IsEnabled	bool	Доступность для взаимодействия	true false	true
IsVisible	bool	Видимость радиокнопки	true false	true
<pre> <RadioButton Source=@ViewModel.IsScalarValueX Name="scalar-value" OnChange=@OnScalarValueXChange Text="Scalar value X" RadioButtonSize=@RadioButtonSize.Small IsVisible=@ViewModel.IsScalar /> <RadioButton Source=@ViewModel.IsScalarValueY Name="scalar-value" OnChange=@OnScalarValueYChange Text="Scalar value Y" RadioButtonSize=@RadioButtonSize.Small IsVisible=@ViewModel.IsScalar /> <RadioButton Source=@ViewModel.IsScalarValueZ Name="scalar-value" OnChange=@OnScalarValueZChange Text="Scalar value Z" RadioButtonSize=@RadioButtonSize.Small IsVisible=@ViewModel.IsScalar /> </pre>				

Внешний вид описанных элементов управления в сравнении с соответствующими стандартными HTML элементами представлен на рисунке 8.

Состояние

	Checkbox	HTML <input> type="checkbox"	RadioButton	HTML <input> type="radio"
Normal	<input type="checkbox"/> Value	<input type="checkbox"/> Value	<input type="radio"/> Value	<input type="radio"/> Value
Hover	<input type="checkbox"/> Value	<input type="checkbox"/> Value	<input type="radio"/> Value	<input type="radio"/> Value
Disabled	<input type="checkbox"/> Value	<input type="checkbox"/> Value	<input type="radio"/> Value	<input type="radio"/> Value
Active	<input checked="" type="checkbox"/> Value	<input checked="" type="checkbox"/> Value	<input checked="" type="radio"/> Value	<input checked="" type="radio"/> Value
Active + Hover	<input checked="" type="checkbox"/> Value	<input checked="" type="checkbox"/> Value	<input checked="" type="radio"/> Value	<input checked="" type="radio"/> Value
Active + Disabled	<input checked="" type="checkbox"/> Value	<input checked="" type="checkbox"/> Value	<input checked="" type="radio"/> Value	<input checked="" type="radio"/> Value

ColorPicker (Normal)

Color =

ColorPicker (Hover)

Color =

ColorPicker (Disabled)

Color =

ColorPicker (Active)

Color =

200

200

20

R

G

B

↕

Размер

<input type="checkbox"/> Value	<input type="checkbox"/> Value	<input type="radio"/> Value	<input type="radio"/> Value
<input checked="" type="checkbox"/> Value	<input checked="" type="checkbox"/> Value	<input checked="" type="radio"/> Value	<input checked="" type="radio"/> Value
Checkbox (Default)	Checkbox (Small)	RadioButton (Default)	RadioButton (Small)

Рисунок 8 – Визуальное представление элементов для выбора значений

3.3. ЭЛЕМЕНТЫ УПРАВЛЕНИЯ ДЛЯ ОТОБРАЖЕНИЯ ДАННЫХ

3.3.1. ЭЛЕМЕНТЫ ДЛЯ ВЫВОДА КОНТЕНТА

Для вывода контента в различных представлениях реализованы следующие типы компонентов:

1. `Label (TitleType = Point)` — элемент управления для отображения текстового контента. Элемент используется в сочетании с элементом управления `PointInput` (таблица 26).

2. `Label (TitleType = Title)` — элемент управления для отображения текстового заголовка. В основном элемент используется в компонентах `Panel` (таблица 27).

3. `Label (TitleType = Default)` — элемент управления для отображения текстового контента. Помимо текстового контента также может выводиться любой другой компонент или тег HTML (таблица 28).

4. `List` — элемент управления для вывода списка элементов. Пользователь может выбрать только один элемент из списка. При выборе значения под списком появляется сообщение с указанием выбранного пользователем элемента (таблица 29).

5. `Table` — элемент управления, представляющий собой таблицу. Строки таблицы могут формироваться и задаваться вручную либо выводиться с помощью цикла. Элемент управления реализован в виде стандартного HTML элемента таблицы с тегом `<table>` (таблица 30).

6. `Row` — строка таблицы. Компонент реализован в виде стандартного HTML элемента с тегом `<tr>`. Элемент является дочерним компонентом по отношению к компоненту `Table` (таблица 31).

7. `Cell` — ячейка таблицы. Компонент реализован в виде стандартного HTML элемента с тегом `<td>`. Элемент является дочерним компонентом по отношению к компоненту `Table` (таблица 32).

Таблица 26 – Label с TitleType = Point (Label)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text (Content)	string	Отображаемый текст (подпись к PointInput)	Любой текст	" "
IsVisible (Visibility)	bool	Видимость текста	true false	true
<code><Label TitleType=@TitleType.Point Text="Arc point:" IsVisible=@ViewModel.IsPointFixed /></code>				

Таблица 27 – Label с TitleType = Title

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text	string	Текст заголовка	Любой текст	" "
ChildContent	RenderFragment	Вложенный контент	Компоненты или теги HTML	null
IsVisible	bool	Видимость текста	true false	true
<code><Label TitleType=@TitleType.Title Text="Harmonic time parameters:" IsVisible=@ViewModel.IsTimeHarmonicMode /></code>				

Таблица 28 – Label с TitleType = Default (Label или TextBlock)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text (Text)	string	Отображаемый текст	Любой текст	" "
ChildContent (Content)	RenderFragment	Вложенный контент	Компоненты или теги HTML	null
OnClick	EventCallback	Обработчик события нажатия клавиши мыши	Callback-функция	null
IsVisible (Visibility)	bool	Видимость текста	true false	true
<code><Label Text="10" IsVisible=@LogScaleX OnClick=@(() => LogScaleX = !LogScaleX) /></code>				

Таблица 29 – List

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Source	IEnumerable<string>	Список значений	Коллекция текстовых строк	null
OnChange	Action<string>	Обработчик события изменения	Callback-функция	null
Selected	string	Выбранное значение	Любой текст	" "
Text	string	Подпись к выбранному значению	Любой текст	" "
IsVisible	bool	Видимость списка	true false	true
<pre><List Source=@FileService.DirectoryFiles Selected=@FileService.Filename OnChange=@((value) => FileService.Filename = value) Text="Selected file: " /></pre>				

Таблица 30 – Table (DataGrid)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
ChildContent	RenderFragment	Вложенный контент	Компоненты или теги HTML	null
IsVisible (Visibility)	bool	Видимость таблицы	true false	true
<pre><Table IsVisible=@(!ViewModel.IsTimeHarmonicMode)> <Row> <Cell Text="Begin time" /> <Cell Text="End time" /> <Cell Text="Internal" /> <Cell Text="Coefficient" /> <Cell Text="Create" /> <Cell Text="Remove" /> </Row> @foreach (var element in ParentViewModel.MeshTimeElements) { <TimeItem ViewModel=@element ParentComponent=@this /> } </Table></pre>				

Таблица 31 – Row (DataGrid.Columns — атрибут DataGrid)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text	string	Текстовый контент	Любой текст	" "
ChildContent	RenderFragment	Вложенный контент	Компоненты или теги HTML	null
OnClick	EventCallback	Обработчик события нажатия клавиши мыши	Callback-функция	null
IsVisible	bool	Видимость строки	true false	true
<pre> <Row> <Cell Text="Begin time" /> <Cell Text="End time" /> <Cell Text="Internal" /> <Cell Text="Coefficient" /> <Cell Text="Layer" /> <Cell Text="Create" /> <Cell Text="Remove" /> </Row> <Row Text="Begin time" /> </pre>				

Таблица 32 – Cell (DataGridTextColumn — атрибут DataGrid)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text	string	Текстовый контент	Любой текст	" "
ChildContent	RenderFragment	Вложенный контент	Компоненты или теги HTML	null
OnClick	EventCallback	Обработчик события нажатия клавиши мыши	Callback-функция	null
<pre> <Cell> <DoubleInput InputType=@InputType.Table Source=@ViewModel.BegT OnChange=@OnBegTChange /> <DoubleInput InputType=@InputType.Table Source=@ViewModel.EndT OnChange=@OnEndTChange /> <IntInput InputType=@InputType.Table Source=@ViewModel.Layer OnChange=@OnLayerChange /> </Cell> <Cell Text="Begin time" /> </pre>				

Внешний вид описанных элементов управления представлен на рисунке 9.

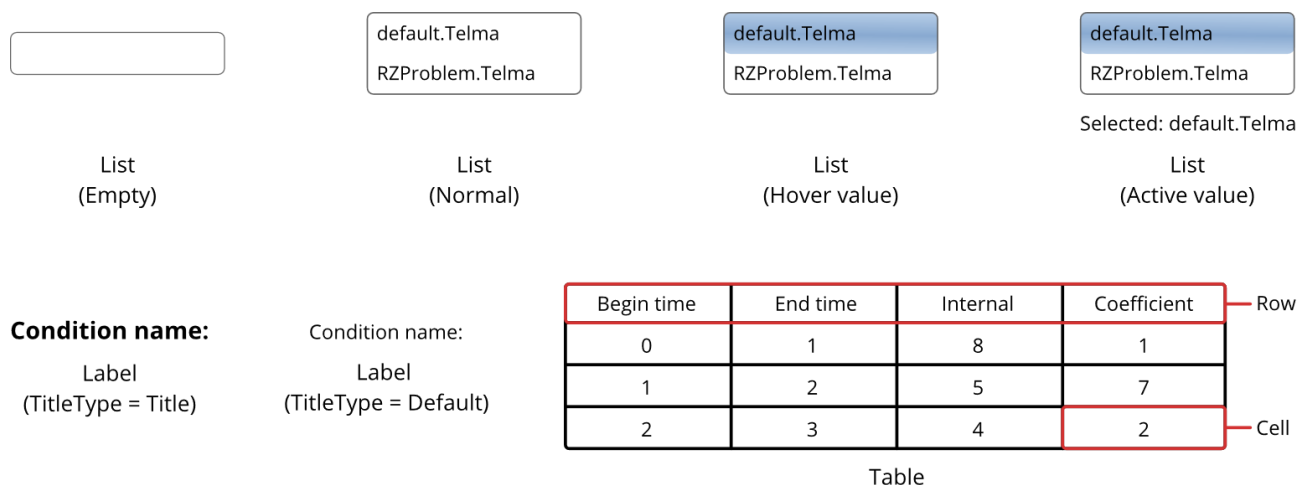


Рисунок 9 – Визуальное представление элементов для вывода контента

3.3.2. ЭЛЕМЕНТЫ СО СКРЫТЫМ КОНТЕНТОМ

Некоторые структурные блоки страницы могут занимать большое количество рабочего пространства, но при этом использоваться относительно редко. Для увеличения рабочей области и визуального упрощения интерфейса используются элементы управления со скрытым контентом. Для скрытия контента реализованы следующие компоненты:

1. `Expander (ExpanderType = Horizontal)` — элемент управления, представляющий собой горизонтальный блок с некоторым контентом. При нажатии на блок содержащийся в нем скрытый контент становится доступным для взаимодействия или исчезает. Скрытый контент появляется ниже блока `Expander` (таблица 33).

2. `Expander (ExpanderType = Vertical)` — элемент управления, представляющий собой вертикальный блок с текстовым заголовком. При нажатии на блок содержащийся в нем скрытый контент становится видимым или исчезает. Скрытый контент появляется справа от блока `Expander` (таблица 34).

3. `FloatingList` — всплывающий список, содержащий внутри себя любые элементы, компоненты и теги HTML (таблица 35).

Таблица 33 – Horizontal Expander (Expander с ExpandDirection="Down")

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Heading (Expander.Header)	RenderFragment	Контент заголовка Expander	Компоненты или теги HTML	null
Content (StackPanel)	RenderFragment	Вложенный в Expander скрытый контент	Компоненты или теги HTML	null
IsExpanded (IsExpanded)	bool	Видимость скрытого контента	true false	false
IsVisible (Visibility)	bool	Видимость Expander	true false	true
<pre> <Expander Expanded="true"> <Heading> <Select Selected=@ViewModel.Name OnChange=@OnMeshChange Source=@ParentViewModel.Project.AvailableMeshes /> </Heading> <Content> <TextInput SizeType=@InputSize.Long IsReadonly="true" Source=@ViewModel.MeshItem.File Text="Mesh file:" /> </Content> </Expander> </pre>				

Таблица 34 – Vertical Expander (Expander с ExpandDirection="Right")

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text	string	Текстовый заголовок	Любой текст	" "
Content (StackPanel)	RenderFragment	Вложенный в Expander скрытый контент	Компоненты или теги HTML	null
IsExpanded (IsExpanded)	bool	Видимость скрытого контента	true false	false
IsVisible (Visibility)	bool	Видимость Expander	true false	true
<pre> <Expander ExpanderType=@ExpanderType.Vertical Text="Modes"> <Content> <ImageButton CanPush="true" Title="Set point selector mode" Image="img/buttons/mode/point.svg" Command=@OnSetPointSelector /> </Content> </Expander> </pre>				

Таблица 35 – FloatingList (ContextMenu)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
<u>Id</u>	string	Уникальный идентификатор списка	Любой текст без пробелов	" "
ChildContent	RenderFragment	Скрытый контент списка	Компоненты или теги HTML	null
IsActive	bool	Видимость списка	true false	false

```
<FloatingList Id="list-curve">
  <ListButton Title="Create hyperbolic segment"
    Text="Hyperbolic"
    Command=@ViewModel.OnCreateHyperbolic />
  <ListButton Title="Create quadrupole hyperbolic segment"
    Text="Quadrupole hyperbolic"
    Command=@ViewModel.OnCreateQuadrupoleHyperbolic />
  <ListButton Title="Create Rp segment"
    Text="Rp segment"
    Command=@ViewModel.OnCreateRfiPolygon
    IsVisible="false" />
  <ListButton Title="Create XY segment"
    Text="XY segment" IsVisible="false"
    Command=@ViewModel.OnCreateXYPolygon />
</FloatingList>
```

Внешний вид описанных элементов управления при различных состояниях представлен на рисунке 10.

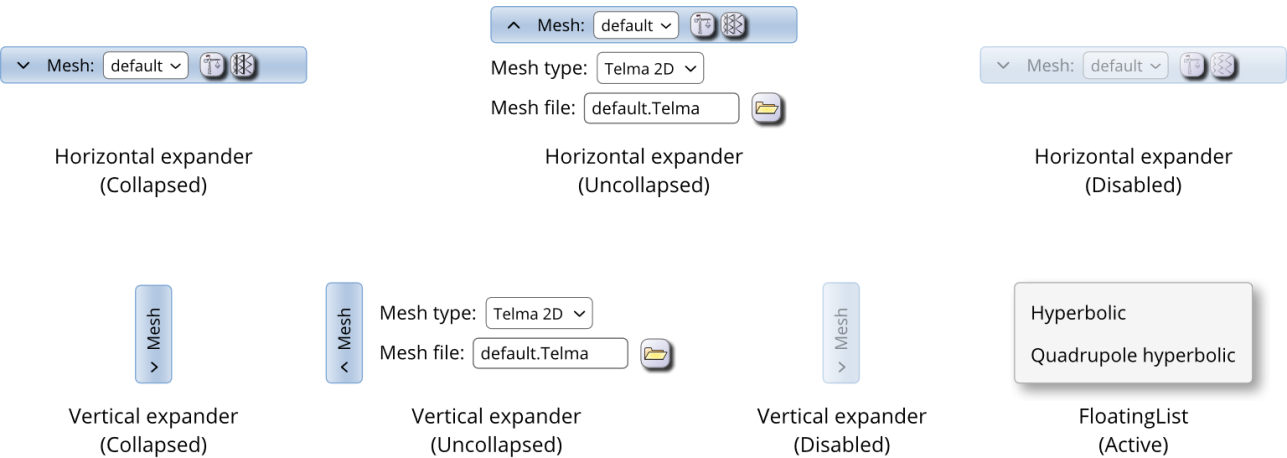


Рисунок 10 – Визуальное представление элементов со скрытым контентом

3.4. КНОПКИ

В приложении Blazor реализовано несколько типов компонентов для кнопок с различным назначением, набором атрибутов, внешним видом, местом использования и поведением:

1. `ImageButton` (`ImageButtonType = Icon`) — кнопка с картинкой в виде иконки, не имеющая состояния зажатия (таблица 36).
2. `ImageButton` (`ImageButtonType = Default`) — кнопка с картинкой, которая может иметь или не иметь состояния зажатия (таблица 37).
3. `ListButton` — кнопка с текстом, которая используется внутри всплывающих списков `FloatingList` (таблица 38).
4. `SectionButton` — кнопка с картинкой и текстом, которая используется в компонентах `Section` (таблица 39).
5. `TextButton` (`TextButtonType = Default`) — кнопка с текстом, которая может иметь или не иметь состояния зажатия. Кнопка имеет различный внешний вид при использовании в компонентах `Panel` и `Section` (таблица 40).
6. `TextButton` (`TextButtonType = Outline`) — кнопка с текстом, имеющая состояние зажатия. Отличается от стандартной кнопки `TextButton` внешним видом (таблица 41).
7. `TextButton` (`TextButtonType = Color`) — кнопка с текстом и окошком для вывода выбранного пользователем цвета. При нажатии на кнопку появляется окно для выбора цвета `ColorPicker`. Кнопка имеет различный внешний вид при использовании в компонентах `Panel` и `Section` (таблица 42).
8. `TextButton` (`TextButtonType = Popup`) — кнопка с текстом, используемая в компонентах `Popup` (таблица 43).
9. `ToggleImageButton` — кнопка с картинкой, которая меняет состояние зажатия на противоположное при нажатии клавишей мыши (таблица 44).
10. `ToggleSectionButton` — кнопка с текстом и картинкой, меняющая состояние на противоположное при нажатии клавишей мыши (таблица 45).

Таблица 36 – ImageButton с ImageButtonType = Icon (Button)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Image (Image)	string	Путь к картинке	Любой текст	""
Title (ToolTip)	string	Всплывающая подсказка	Любой текст	""
Class	string	Список CSS-классов	Любой текст	""
OnClickEvent	EventCallback	Обработчик события нажатия клавиши мыши	Callback-функция	null
IsVisible (Visibility)	bool	Видимость кнопки	true false	true
<code><ImageButton ButtonType=@ImageButtonType.Icon Class="modify__change" Title="Change element name" OnClickEvent=@ChangeElement Image="img/check.svg" IsVisible=@IsEditButtonVisible /></code>				

Таблица 37 – ImageButton (Button со стилем TelmaButton)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Image (Image)	string	Путь к картинке	Любой текст	""
Title (ToolTip)	string	Всплывающая подсказка	Любой текст	""
Command (Command)	ITelmaCommand	Действие для кнопки	TelmaCommand	null
ButtonSize	Enum ButtonSize	Размер кнопки	Tiny Small Middle Big	Big
CanPush	bool	Наличие зажатия	true false	false
IsPushed (IsChecked)	bool	Состояние зажатия	true false	false
IsVisible (Visibility)	bool	Видимость кнопки	true false	true
<code><ImageButton CanPush="true" Title="Create line segment (Ctrl + I)" Image="img/buttons/main-form/Create/CreateLineSegment.svg" Command=@ViewModel.OnCreateLine IsPushed=@ViewModel.IsAddLineMode /></code>				

Таблица 38 – ListButton (MenuItem в ContextMenu)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text	string	Текст внутри кнопки	Любой текст	""
Title (ToolTip)	string	Всплывающая подсказка	Любой текст	""
Command (Command)	ITelmaCommand	Действие для кнопки	TelmaCommand	null
ButtonType	Enum ListButtonType	Тип кнопки	Underline Default	Default
CanPush	bool	Наличие зажатия	true false	false
IsPushed (IsChecked)	bool	Состояние зажатия	true false	false
IsVisible (Visibility)	bool	Видимость кнопки	true false	true
<pre><ListButton Title="Copy selected by vector" Text="Copy by vector" Command=@ViewModel.OnCopyVector /></pre>				

Таблица 39 – SectionButton (RibbonButton)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text (Caption)	string	Текст в первой строке	Любой текст	""
Text2Row	string	Текст во второй строке	Любой текст	""
Image (Image)	string	Путь к картинке	Любой текст	""
Title (ToolTip)	string	Всплывающая подсказка	Любой текст	""
Command (Command)	ITelmaCommand	Действие для кнопки	TelmaCommand	null
CanPush	bool	Наличие зажатия	true false	false
IsPushed (IsChecked)	bool	Состояние зажатия	true false	false
IsVisible (Visibility)	bool	Видимость кнопки	true false	true
<pre><SectionButton Title="Remove node" Text="Remove" Text2Row="node" Image="img/buttons/base-plane/RemoveNode.svg" Command=@ViewModel.OnRemoveNode /></pre>				

Таблица 40 – TextButton с TextButtonType = Default (Button или RibbonButton)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text (Caption)	string	Текст внутри кнопки	Любой текст	""
Title (ToolTip)	string	Всплывающая подсказка	Любой текст	""
Command (Command)	ITelmaCommand	Действие для кнопки	TelmaCommand	null
CanPush	bool	Наличие зажатия	true false	false
IsPushed	bool	Состояние зажатия	true false	false
IsVisible (Visibility)	bool	Видимость кнопки	true false	true
<pre><TextButton CanPush="true" Title="Move selected by vector" Text="Move by vector" Command=@ViewModel.OnMoveVector IsPushed=@ViewModel.IsMoveByVector /></pre>				

Таблица 41 – TextButton с TextButtonType = Outline (Стиль TelmaTextButton)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text (Caption)	string	Текст внутри кнопки	Любой текст	""
Title (ToolTip)	string	Всплывающая подсказка	Любой текст	""
Command (Command)	ITelmaCommand	Действие для кнопки	TelmaCommand	null
CanPush	bool	Наличие зажатия	true false	false
IsPushed	bool	Состояние зажатия	true false	false
IsVisible (Visibility)	bool	Видимость кнопки	true false	true
<pre><TextButton ButtonType=@TextButtonType.Outline Text="Start calculation" Command=@ViewModel.StartCalculation /></pre>				

Таблица 42 – TextButton с TextButtonType = Color (Button + Background binding)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text (Caption)	string	Текст внутри кнопки	Любой текст	""
Title (ToolTip)	string	Всплывающая подсказка	Любой текст	""
<u>Id</u>	string	Уникальный идентификатор кнопки	Любой текст без пробелов	""
Source (Background)	Color	Выбранный цвет	Любой цвет	#ffffff (белый)
Command (Command)	ITelmaCommand	Действие для кнопки	TelmaCommand	null
IsVisible (Visibility)	bool	Видимость кнопки	true false	true
<pre><TextButton ButtonType=@TextButtonType.Color Id="text-color" Text="Text color" Command=@ViewModel.OnTextColor Source=@ViewModel.ActiveTitleView.TextColor Title="Text color" /></pre>				

Таблица 43 – TextButton с TextButtonType = Popup (Кнопки в MessageBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text (Caption)	string	Текст внутри кнопки	Любой текст	""
Title (ToolTip)	string	Всплывающая подсказка	Любой текст	""
OnClickEvent	EventCallback	Обработчик события нажатия клавиши мыши	Callback-функция	null
IsVisible (Visibility)	bool	Видимость кнопки	true false	true
<pre><TextButton ButtonType=@TextButtonType.Popup Text="Cancel" OnClickEvent=@(() => Cancel()) /></pre>				

Таблица 44 – ToggleImageButton (CheckBox со стилем TelmaToggleButton)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Image (Image)	string	Путь к картинке	Любой текст	""
Title (ToolTip)	string	Всплывающая подсказка	Любой текст	""
Command (Command)	Telma-CheckCommand	Действие для кнопки	Telma-CheckCommand	null
ButtonSize	Enum ButtonSize	Размер кнопки	Tiny Small Middle Big	Big
IsPushed (IsChecked)	bool	Состояние зажатия	true false	false
IsVisible (Visibility)	bool	Видимость кнопки	true false	true
<pre><ToggleButton Title="Color view" Command=@ViewModel.FilledDraw Image="img/buttons/main-form/Change/Ink.svg" IsPushed=@(ViewModel.FilledDraw.IsChecked == true) /></pre>				

Таблица 45 – ToggleSectionButton (RibbonToggleButton)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Text (Caption)	string	Текст в первой строке	Любой текст	""
Text2Row	string	Текст во второй строке	Любой текст	""
Image (Image)	string	Путь к картинке	Любой текст	""
Title (ToolTip)	string	Всплывающая подсказка	Любой текст	""
Command (Command)	Telma-CheckCommand	Действие для кнопки	Telma-CheckCommand	null
IsPushed (IsChecked)	bool	Состояние зажатия	true false	false
IsVisible (Visibility)	bool	Видимость кнопки	true false	true
<pre><ToggleSectionButton Title="Set add title mode" Text="Add titles mode" Image="img/ AddTitlesMode.svg" Command=@ViewModel.OnTitleSetAddMode IsPushed=@(ViewModel.OnTitleSetAddMode.IsChecked == true) /></pre>				

Внешний вид кнопок при различных состояниях представлен на рисунке 11.

	Normal	Hover	Active (Pushed)	Disabled
ListButton (Default)				
ListButton (Underline)				
Default TextButton (In Section)	Move by vector			Move by vector
Default TextButton (In Panel)				
Outline TextButton				
Color TextButton (In Section)	Fill color			Fill color
Color TextButton (In Panel)				
Popup TextButton				
SectionButton (ToggleSectionButton)				
Big ImageButton (ToggleImageButton)				
Middle ImageButton (ToggleImageButton)				
Small ImageButton (ToggleImageButton)				
Tiny ImageButton (ToggleImageButton)				
Icon ImageButton	✓			✓

Рисунок 11 – Визуальное представление кнопок

3.5. КОМПОНЕНТЫ ДЛЯ ПОСТРОЕНИЯ СТРАНИЦ

3.5.1. ПАНЕЛИ

Панели в интерфейсе Telma содержат большую часть элементов управления, необходимых для взаимодействия пользователя с программным комплексом в процессе решения задачи.

Панели в боковой части экрана расположены вертикально и последовательно. Панели в верхней части экрана расположены горизонтально и реализованы в виде табов — вкладок, переключаемых по нажатию клавиши мыши. Одновременно могут быть активны несколько боковых панелей и лишь одна из верхних панелей.

В Telma панели с одинаковыми функциями в боковой и верхней частях экрана являются одним компонентом с разным визуальным представлением в зависимости от места вывода на экран. В приложении Blazor реализуем верхние и боковые панели в виде различных компонентов.

В платформе WPF табы являются стандартными элементами управления. Поскольку язык HTML не предоставляет стандартных средств для создания табов, реализуем их функциональность самостоятельно. Для добавления панелей реализованы следующие типы компонентов:

1. `Panel` — компонент, используемый для добавления боковых панелей (таблица 46).

2. `TabControl` — компонент, используемый для добавления и переключения верхних панелей (таблица 47).

3. `TabPage` — вкладка, представляющая собой верхнюю панель. Элемент является дочерним компонентом по отношению к компоненту `TabControl` (таблица 48).

4. `Section` — группа элементов управления интерфейса, объединенных по назначению. Элемент является дочерним компонентом по отношению к компоненту `TabPage` (таблица 49).

Таблица 46 – Panel

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Title	string	Название панели	Любой текст	""
<u>Id</u>	string	Уникальный идентификатор панели	Любой текст без пробелов	""
ChildContent	RenderFragment	Содержимое панели	Компоненты или теги HTML	null
IsTitleVisible	bool	Видимость заголовка панели	true false	true
IsVisible (Visibility)	bool	Видимость панели	true false	true
<pre> <Panel Id="task" Title="Task panel"> <Expander> <Heading>Problems</Heading> <Content> @foreach (var problem in ViewModel.AvailableProblems) { <ProblemItem ViewModel=@problem /> } </Content> </Expander> </Panel> </pre>				

Таблица 47 – TabControl (RibbonTab)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
ActivePageName	string	Имя активной вкладки	Любой текст	""
ChildContent	RenderFragment	Верхняя панель в виде вкладки	Компоненты или теги HTML	null
<pre> <TabControl ActivePageName="Edit"> @foreach (var pair in upperPanels) { <TabPage Title=@pair.Key> @foreach (var panel in pair.Value) { @RenderPanel (panel.panelType, panel.viewModelType) } </TabPage> } </TabControl> </pre>				

Таблица 48 – TabPage

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Title	string	Название вкладки	Любой текст	""
ChildContent	RenderFragment	Секции верхней панели	Компоненты или теги HTML	null
<pre><TabPage Title=@pair.Key> @foreach (var panel in pair.Value) { @RenderPanel(panel.panelType, panel.viewModelType) } </TabPage></pre>				

Таблица 49 – Section (RibbonGroup)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Title	string	Название секции	Любой текст	""
ChildContent	RenderFragment	Содержимое секции	Компоненты или теги HTML	null
SectionType	Enum SectionType	Тип секции	Text Default	Default
IsVisible (Visibility)	bool	Видимость секции	true false	true
<pre><Section Title="Copy" SectionType=@SectionType.Text> <TextButton Title="Copy selected by vector" Text="Copy by vector" Command=@ViewModel.OnCopyVector /> <TextButton Title="Copy selected around point" Text="Copy around point" Command=@ViewModel.OnCopyAroundPoint /> <TextButton Title="Copy selected around zero" Text="Copy around zero" Command=@ViewModel.OnCopyAroundZero /> <TextButton Title="Copy selected by reflection" Text="Copy by reflection" Command=@ViewModel.OnCopyReflection /> <TextButton Title="Copy selected by scale" Text="Copy by scale" Command=@ViewModel.OnCopyScale /> </Section></pre>				

Компоненты панелей добавляются в разметку страницы главного окна приложения динамически. Функция добавления панелей с указанием необходимой ViewModel представлена на рисунке 12.

```
RenderFragment RenderPanel(Type PanelType, Type ViewModelType)
{
    RenderFragment RenderFragment = Builder =>
    {
        Builder.OpenComponent(0, PanelType);

        switch (ViewModelType.Name)
        {
            case "TelmaComponent":
                Builder.AddAttribute(1, "ViewModel", ViewModel);
                break;
            case "TelbaseComponent":
                Builder.AddAttribute(1, "ViewModel",
                                    (TelbaseComponent)ViewModel);
                break;
            case "PostProcessorComponent":
                Builder.AddAttribute(1, "ViewModel",
                                    (PostProcessorComponent)ViewModel);
                break;
            case "ProcessorManagerComponent":
                Builder.AddAttribute(1, "ViewModel",
                                    (ProcessorManagerComponent)ViewModel);
                break;
            case "TelmaComponentUIAggregator":
                Builder.AddAttribute(1, "ViewModel", aggregator);
                break;
            default:
                break;
        }

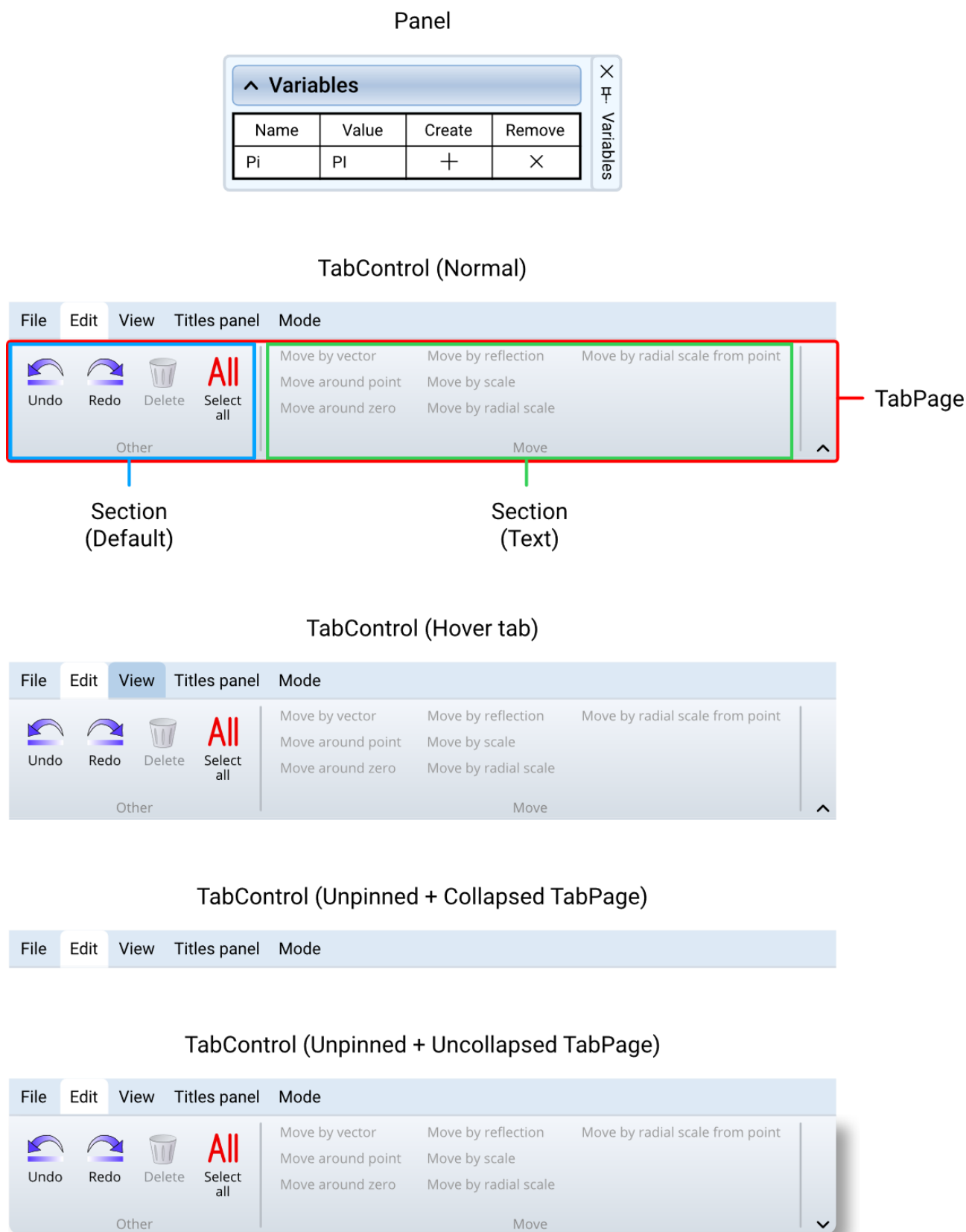
        if (PanelType == typeof(SelectionPanel) ||
            PanelType == typeof(TelbasePanel) ||
            PanelType == typeof(OtherSection) ||
            PanelType == typeof(WorkModeSection) ||
            PanelType == typeof(SelectorSection) ||
            PanelType == typeof(SelectionMethodSection) ||
            PanelType == typeof(MouseModeSection))
            Builder.AddAttribute(2, "Aggregator", aggregator);

        Builder.CloseComponent();
    };

    return RenderFragment;
}
```

Рисунок 12 – Функция для добавления панелей в разметку окна

Внешний вид описанных панелей представлен на рисунке 13.



3.5.2. ОКНА ДЛЯ ВЗАИМОДЕЙСТВИЯ С ПОЛЬЗОВАТЕЛЕМ

Для взаимодействия с пользователем посредством различных окон реализованы следующие компоненты:

1. `Parameters` — окно для задания и отображения параметров объектов в препроцессоре. Окна с параметрами загружаются динамически при рисовании объектов или их выборе нажатием клавиши мыши (таблица 50).

2. `Popup` — всплывающее окно для интерактивного взаимодействия с пользователем и запроса данных (таблица 51).

Таблица 50 – `Parameters (BaseWrapperPanel)`

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
<code>Title</code>	<code>string</code>	Название окна	Любой текст	<code>""</code>
<code>Id</code>	<code>string</code>	Уникальный идентификатор окна	Любой текст без пробелов	<code>""</code>
<code>ChildContent</code>	<code>RenderFragment</code>	Содержимое окна	Компоненты или теги HTML	<code>null</code>
<pre><Parameters Title="Boundaries edit" Id="boundaries-edit"> <Wrapper> <Checkbox Text="Coefficient:" Source=@ViewModel.IsCoeff Id="boundaries-coeff" OnChange=@OnCoeffCheckboxChange /> <DoubleInput SizeType=@InputSize.Short Source=@ViewModel.Coeff OnChange=@OnCoeffChange /> </Wrapper> <Wrapper> <Checkbox Input Text="Internal:" Source=@ViewModel.IsInternal Id="boundaries-internal" OnChange=@OnInternalCheckboxChange /> <IntInput SizeType=@InputSize.Short Source=@ViewModel.Internal OnChange=@OnInternalChange /> </Wrapper> <Wrapper> <Checkbox CheckboxType=@CheckboxType.Input Text="Condition:" Id="boundaries-condition" OnChange=@OnConditionCheckboxChange Source=@ViewModel.IsCondition /> <Select Selected=@ViewModel.ConditionName Source=@BoundaryNames OnChange=@OnBoundaryChange /> </Wrapper> </Parameters></pre>				

Таблица 51 – Popup (MessageBox)

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
Title (Caption)	string	Название окна	Любой текст	""
<u>I</u> d	string	Уникальный идентификатор окна	Любой текст без пробелов	""
ChildContent	RenderFragment	Содержимое окна	Компоненты или теги HTML	null

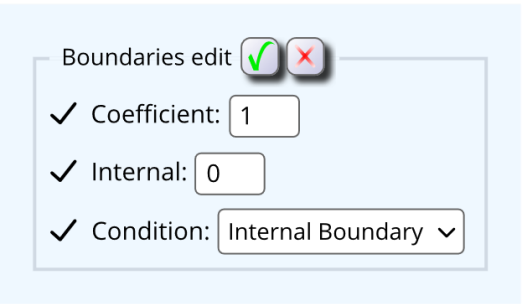
```
<Popup Id="load-name" Title="File directory">
  <Label Text="Choose a file:" />

  <List Source=@FileService.DirectoryFiles
    OnChange=@((value) => FileService.Filename = value)
    Selected=@FileService.Filename
    Text="Selected file: " />

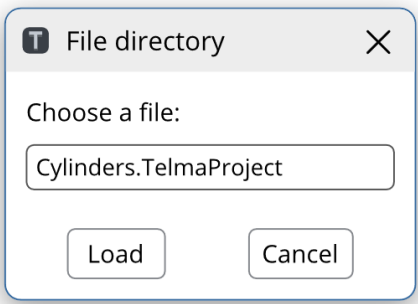
  <Container Classes="popup_buttons">
    <TextButton ButtonType=@TextButtonType.Popup
      Text="Load"
      OnClickEvent=@(() => Load()) />

    <TextButton ButtonType=@TextButtonType.Popup
      Text="Cancel"
      OnClickEvent=@(() =>
        await JSInterop.InvokeVoidAsync("ClosePopup", "load-name")) />
  </Container>
</Popup>
```

Внешний вид описанных окон представлен на рисунке 14.



Parameters



Popup

Рисунок 14 – Визуальное представление окон

3.6. ОБЕРТКИ ДЛЯ ЭЛЕМЕНТОВ ИНТЕРФЕЙСА

Компоненты-обертки не имеют визуального представления. В приложении Blazor реализованы следующие типы компонентов-обертки:

1. `FormButtons` — обертка для группировки наборов кнопок в боковых панелях. С помощью `FormButtons` кнопки располагаются по горизонтали на одинаковом расстоянии друг от друга и центрируются внутри обертки по вертикали. Кроме того, обертка ограничивает максимальную ширину набора кнопок, позволяет задать всем кнопкам набора одинаковый цвет фона и задает отступ до следующего блока (таблица 52).

2. `Wrapper` — внешняя обертка для группировки различных элементов управления. С помощью этого компонента вложенные элементы располагаются по горизонтали и центрируются внутри обертки по вертикали. Если группу кнопок внутри обертки `FormButtons` необходимо разбить на подгруппы, то каждую такую подгруппу следует обернуть в компонент `Wrapper`. Компонент `Wrapper` с типом `Point` следует использовать для `PointInput`. Блоки `Wrapper` могут использоваться в любом месте приложения. Чтобы расположить эти блоки по горизонтали, их следует вложить внутрь `Wrapper` (таблица 53).

3. `FormInner` — внутренняя обертка для частей контента боковой панели. С помощью этого компонента вложенные блоки также располагаются по горизонтали и центрируются внутри обертки по вертикали. Компонент `FormInner` задает для вложенного контента внешние отступы сверху и снизу. Если `FormInner` находится в начале панели, то он не имеет отступа сверху, а если в конце — не имеет отступа снизу. Указав для `FormInner` тип `Grid`, можно расположить элементы в две колонки на одинаковом расстоянии. Обертки `FormInner` используются только внутри боковых панелей (таблица 54).

4. `Container` — универсальная обертка, не имеющая никаких CSS-стилей. С помощью атрибута `Classes` компоненту задаются любые CSS классы, которые впоследствии стилизуются во внешнем файле CSS (таблица 55).

Таблица 52 – FormButtons

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
ChildContent	RenderFragment	Содержимое блока обертки	Компоненты или теги HTML	null
ButtonsColor	Enum ButtonsColor	Цвет фона для набора кнопок внутри обертки	Green Yellow Red Orange Blue Violet	Violet
IsVisible	bool	Видимость блока обертки	true false	true
<pre> <FormButtons ButtonsColor=@ButtonsColor.Red> <ImageButton Title="Set max step" Image="img/buttons/main-form/Build/SetMaxStep.svg" Command=@ViewModel.OnSetBoundaryMaxStep /> <ImageButton Title="Set min step" Image="img/buttons/main-form/Build/SetMinStep.svg" Command=@ViewModel.OnSetBoundaryMinStep /> <ImageButton Title="Set equal max step" Image="img/buttons/main-form/Build/GE.svg" Command=@ViewModel.OnSetBoundaryEqualMaxNode /> </FormButtons> </pre>				

Таблица 53 – Wrapper

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
ChildContent	RenderFragment	Содержимое блока обертки	Компоненты или теги HTML	null
WrapperType	Enum WrapperType	Тип блока обертки	Point Default	Default
IsVisible	bool	Видимость блока обертки	true false	true
<pre> <Wrapper WrapperType=@WrapperType.Point> <Checkbox OnChange=@OnFixStartChange Id="arc-fix-start" Text="Fix start point:" Source=@ViewModel.FixedBeg /> <PointInput Source=@ViewModel.BegPoint.As3D() OnChange=@OnBegChange /> </Wrapper> </pre>				

Таблица 54 – FormInner

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
ChildContent	RenderFragment	Содержимое блока обертки	Компоненты или теги HTML	null
FormInnerType	Enum FormInnerType	Тип блока обертки	Grid Flex	Flex
IsVisible	bool	Видимость блока обертки	true false	true
<pre> <FormInner FormInnerType=@FormInnerType.Grid IsVisible=@(!ViewModel.IsTimeHarmonicMode)> <DoubleInput Text="Start time:" SizeType=@InputSize.Middle Source=@ViewModel.StartTime OnChange=@OnStartTimeChange /> <DoubleInput Text="Stop time:" SizeType=@InputSize.Middle Source=@ViewModel.StopTime OnChange=@OnStopTimeChange /> <IntInput Text="Skip calc:" SizeType=@InputSize.Middle Min="0" Source=@ViewModel.TimeCalcSkip OnChange=@OnSkipCalcChange /> <IntInput Text="Skip save:" SizeType=@InputSize.Middle Min="0" Source=@ViewModel.TimeSaveSkip OnChange=@OnSkipSaveChange /> <IntInput Text="Skip res:" SizeType=@InputSize.Middle Min="0" Source=@ViewModel.TimeCalcResSkip OnChange=@OnSkipResChange /> <IntInput Text="Layers:" SizeType=@InputSize.Middle Min="2" Max="4" Source=@ViewModel.TimeLayers OnChange=@OnLayersChange /> </FormInner> </pre>				

Таблица 55 – Container

Название атрибута	Тип значения	Описание атрибута	Допустимые значения	Значение по умолчанию
ChildContent	RenderFragment	Содержимое блока обертки	Компоненты или теги HTML	null
Classes	string	Набор CSS-классов	Любой набор слов	""
Style	string	Набор CSS-свойств	Любые CSS-свойства	""
IsVisible	bool	Видимость блока обертки	true false	true
<pre> <Container Classes="popup_buttons"> <TextButton ButtonType=@TextButtonType.Popup Text="Load" OnClickEvent=@(() => Load()) /> <TextButton ButtonType=@TextButtonType.Popup Text="Cancel" OnClickEvent=@(() => Cancel()) /> </Container> </pre>				

3.7. УПРОЩЕНИЕ И ОПТИМИЗАЦИЯ ПРОЦЕССА РАЗРАБОТКИ

Элементы управления HTML по умолчанию имеют различный внешний вид во всех современных браузерах, что приводит к необходимости доработки веб-интерфейсов под каждый браузер в отдельности. Это обстоятельство влечет за собой дополнительные временные и материальные затраты, которых можно избежать при грамотном подходе к процессу разработки.

Наиболее эффективным решением данной проблемы является использование специальных библиотек CSS для приведения стилей элементов к единому виду. Устранение различий в отображении достигается путем сброса или переопределения стилей, заданных в браузерах по умолчанию. Поскольку некоторые стандартные стили для элементов являются полезными и не нуждаются в удалении, рекомендуется переопределять только необходимые CSS-свойства, а не сбрасывать все стили, установленные браузерами.

В данной работе для обеспечения кроссбраузерности стилей использовалась библиотека `normalize.css`. Данная библиотека представляет собой настраиваемый файл с набором CSS-свойств, переопределяющий внешний вид некоторых элементов в соответствии с современными стандартами. Нормализация базовых стилей для большинства элементов HTML в сочетании с самостоятельной разработкой элементов управления нивелирует различия в конечном виде пользовательского интерфейса в разных браузерах и избавляет от необходимости его доработки.

Помимо проблемы совместимости стилей стандартный CSS имеет ряд ограничений, замедляющих процесс разработки веб-приложений. Одним из способов преодоления этих ограничений является использование расширений языка CSS с помощью различных препроцессоров. Наиболее популярными из них являются SASS, SCSS, LESS и Stylus. В данной работе использовался препроцессор SASS, поэтому рассмотрим его возможности по оптимизации процесса разработки более подробно.

SASS (Syntactically Awesome Stylesheets) представляет собой метаязык на основе языка CSS, созданный с целью упрощения написания стилей для оформления HTML элементов. Синтаксис языка SASS отличается от синтаксиса CSS отсутствием фигурных скобок и точек с запятой.

Важным преимуществом SASS является поддержка вложенности селекторов, недоступная в стандартном CSS. Вложенность элементов в SASS реализована с помощью отступов. Другой особенностью SASS является возможность создания переменных для хранения многократно используемых значений CSS-свойств, что позволяет при необходимости вносить правки в единственном месте приложения. Кроме переменных препроцессор поддерживает использование циклов, массивов, условных конструкций и других инструментов.

Для получения CSS-файла следует скомпилировать файл со стилями, написанными на языке SASS. Компиляция может выполняться на стороне сервера или клиента в момент загрузки веб-страницы либо с помощью сборщиков проектов и сторонних приложений во время разработки приложения. В данной работе для компиляции SASS в CSS использовался сборщик проектов Gulp.

Gulp представляет собой программу на языке JavaScript в виде набора инструментов и плагинов для автоматизации повторяющихся задач и оптимизации рабочего процесса с целью повышения эффективности разработки. Помимо компиляции SASS-файлов Gulp решает и другие важные задачи:

1. Слежение за изменениями в файлах HTML, CSS и JavaScript и автоматическое обновление веб-страницы при сохранении кода в них. Это позволяет моментально видеть примененные изменения аналогично работе с WPF.
2. Автоматическое добавление вендорных префиксов CSS-свойствам для их использования в различных браузерах.
3. Компиляция кода на языке TypeScript в код на языке JavaScript.
4. Минификация файлов с JavaScript кодом и CSS стилями.
5. Объединение файлов библиотек и файлов проекта в единый файл.
6. Сборка файлов проекта в финальную версию без включения лишнего содержимого.

4. РЕАЛИЗОВАННЫЙ ФУНКЦИОНАЛ MVVM VIEW

4.1. ВНЕШНИЙ ВИД СТРАНИЦ ПРИЛОЖЕНИЯ

Внешний вид страниц для модулей Telma представлен на рисунках 15–20.

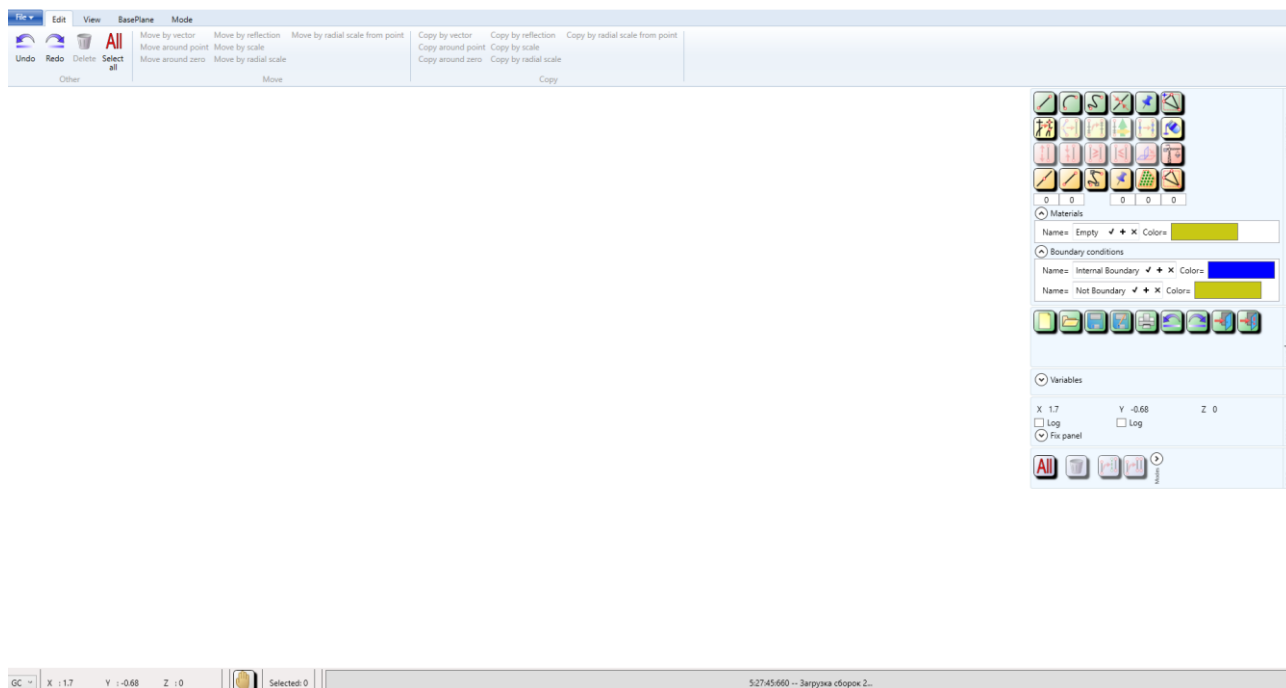


Рисунок 15 – Внешний вид страницы препроцессора в Telma

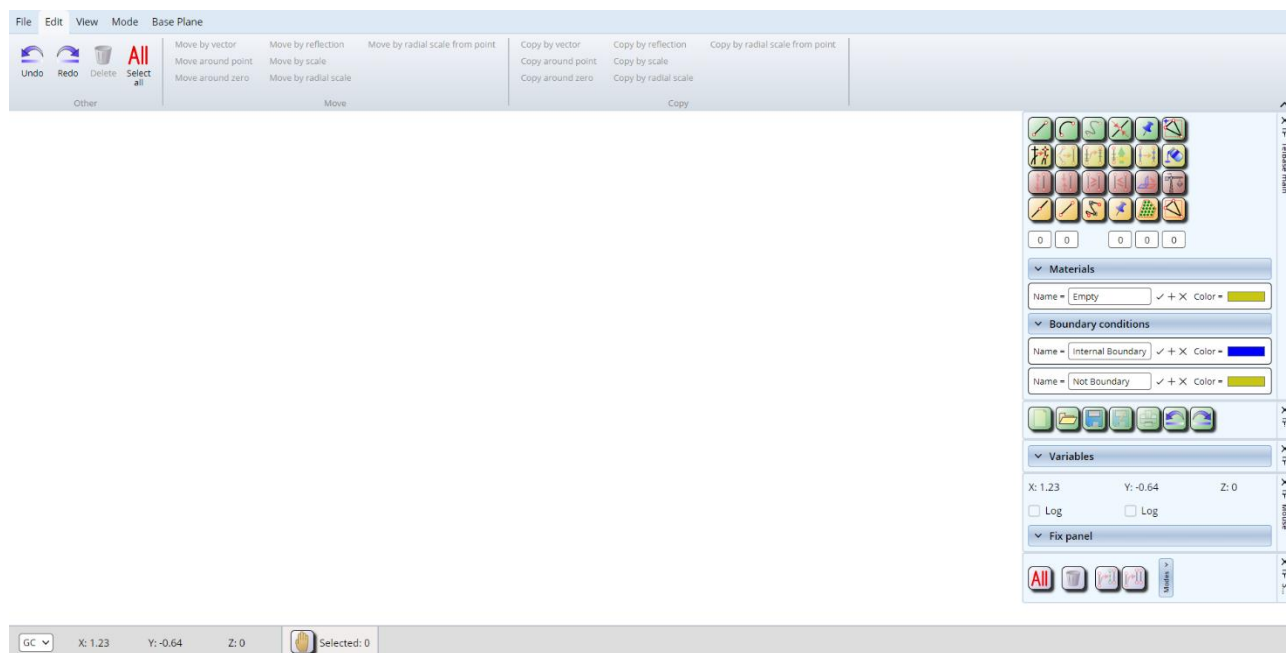


Рисунок 16 – Внешний вид страницы препроцессора в приложении Blazor

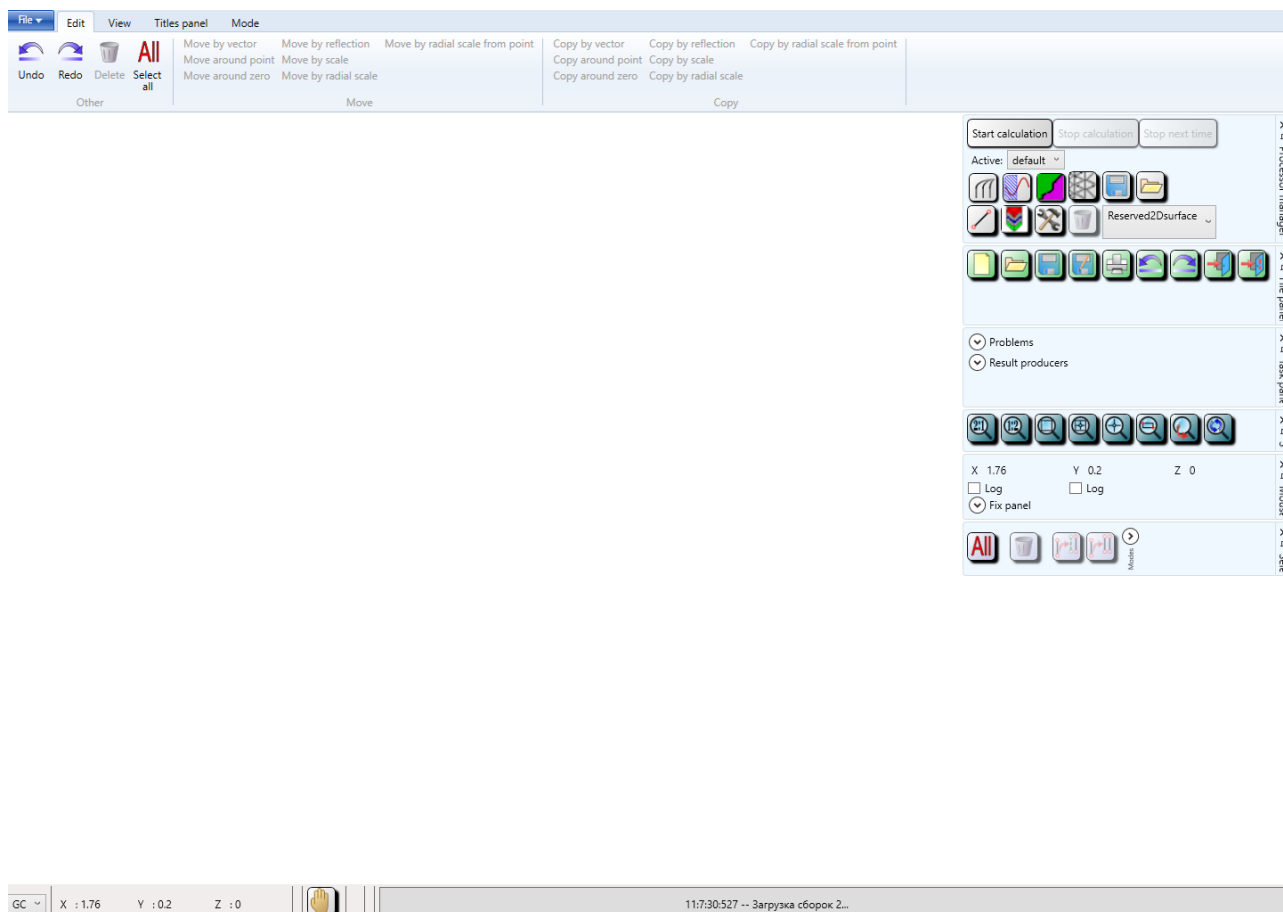


Рисунок 17 – Внешний вид страницы процессора в Telma

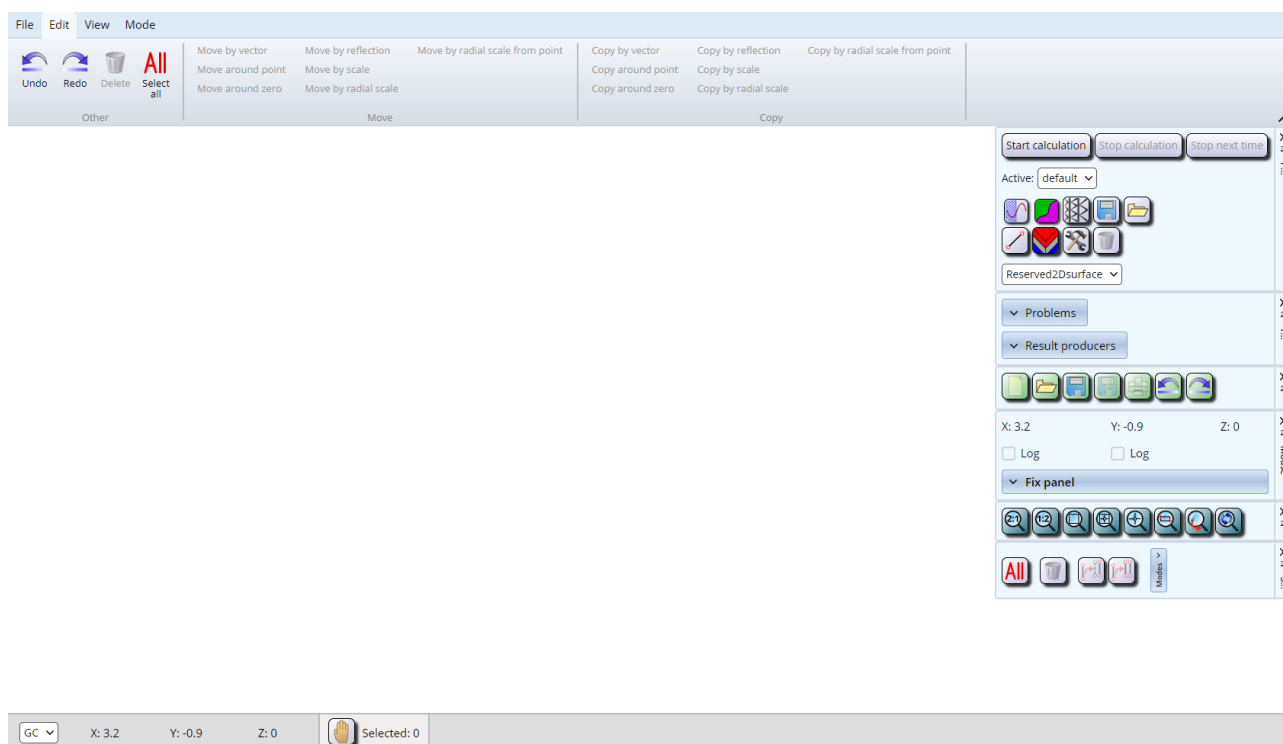


Рисунок 18 – Внешний вид страницы процессора в приложении Blazor

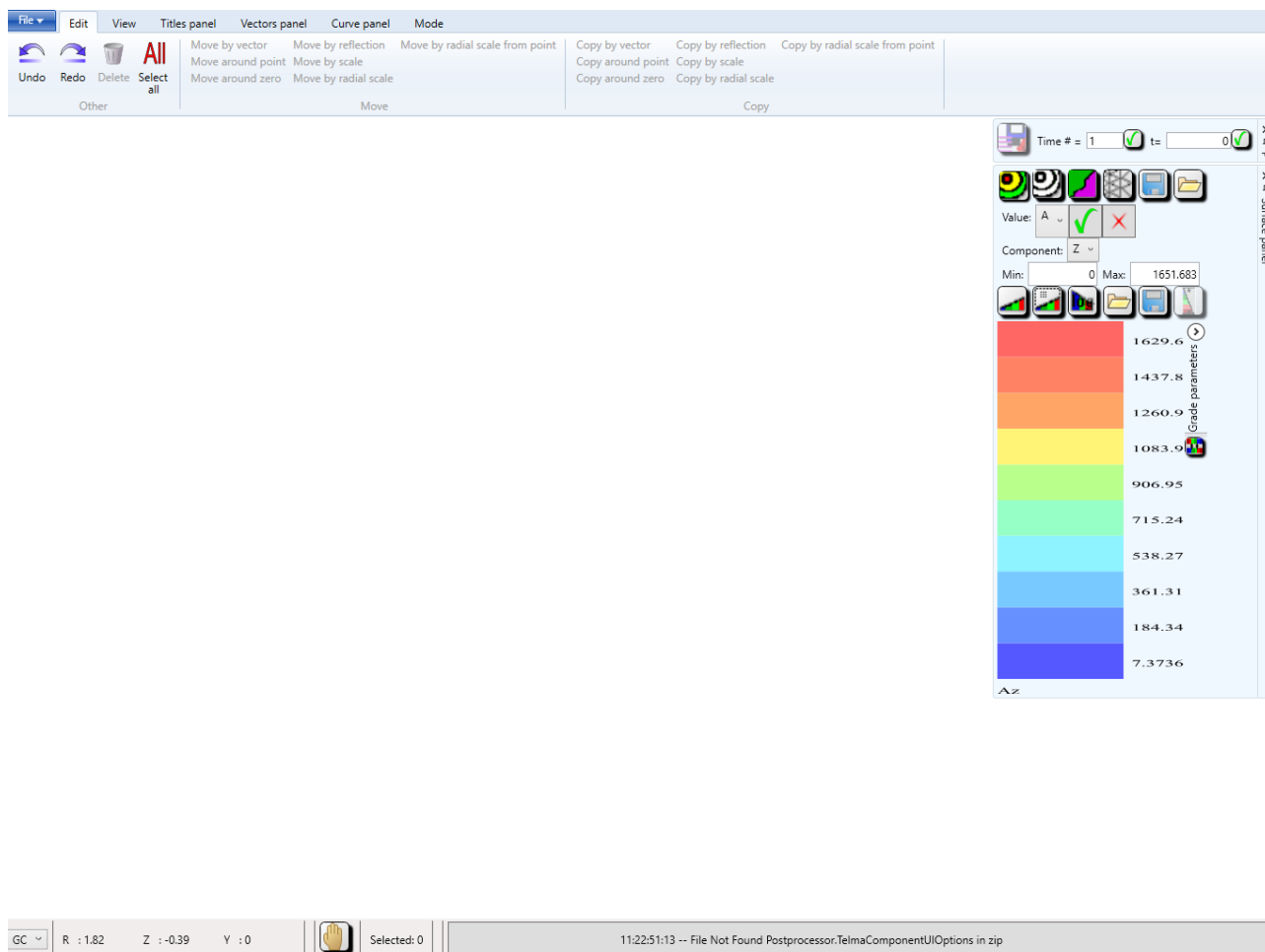


Рисунок 19 – Внешний вид страницы постпроцессора в Telma

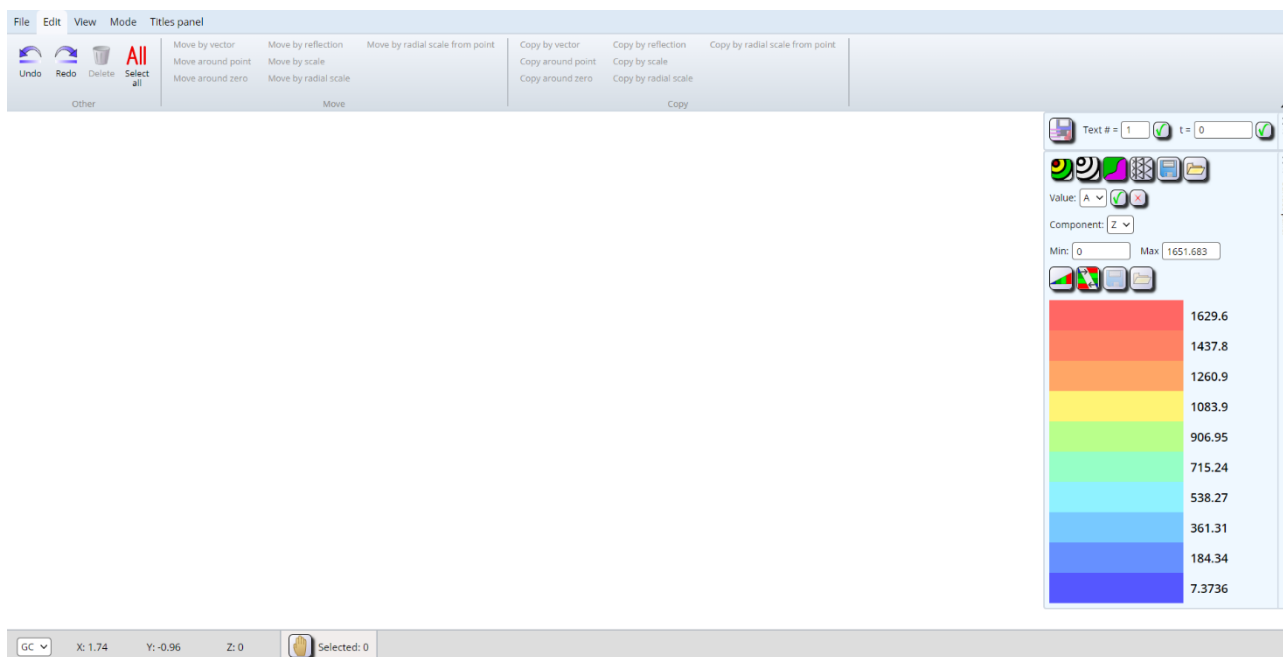


Рисунок 20 – Внешний вид страницы постпроцессора в приложении Blazor

4.2. ПАНЕЛИ В ВЕРХНЕЙ ЧАСТИ ЭКРАНА

На рисунках 21–30 изображены панели в верхней части экрана.

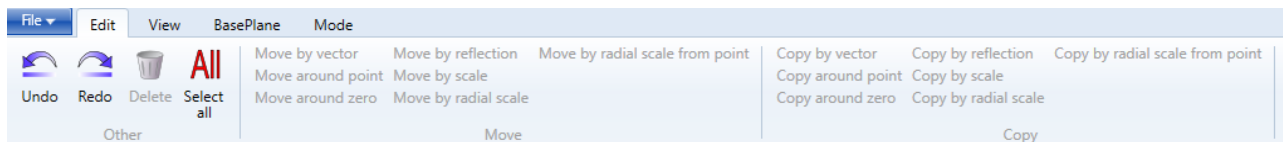


Рисунок 21 – Панель Edit в Telma



Рисунок 22 – Панель Edit в веб-версии

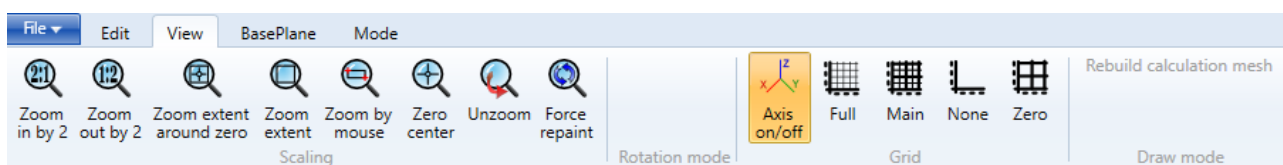


Рисунок 23 – Панель View в Telma

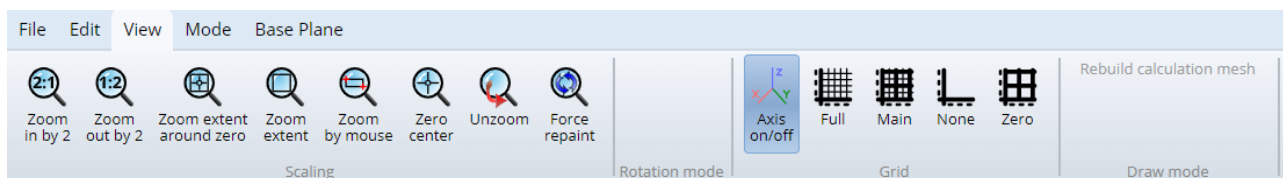


Рисунок 24 – Панель View в веб-версии

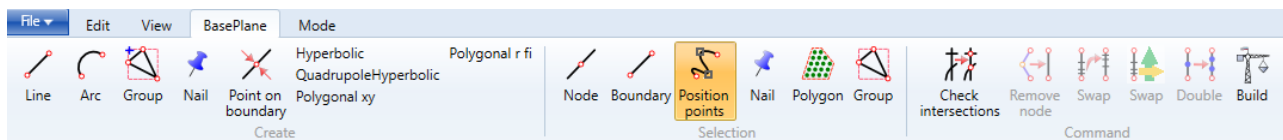


Рисунок 25 – Панель Base plane в Telma

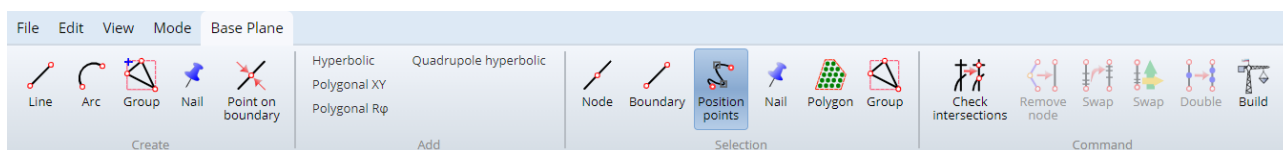


Рисунок 26 – Панель Base plane в веб-версии

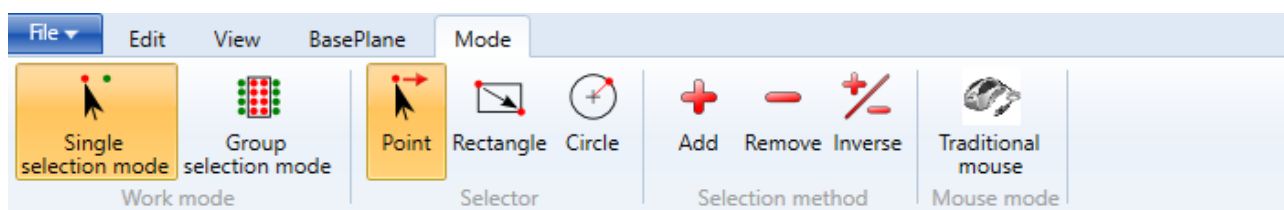


Рисунок 27 – Панель Mode в Telma

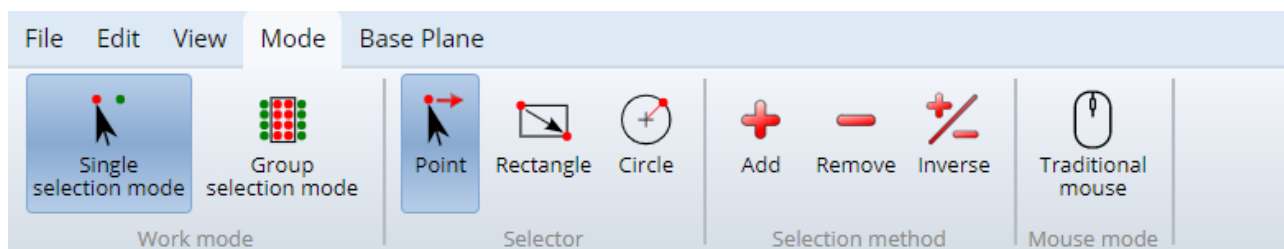


Рисунок 28 – Панель Mode в веб-версии

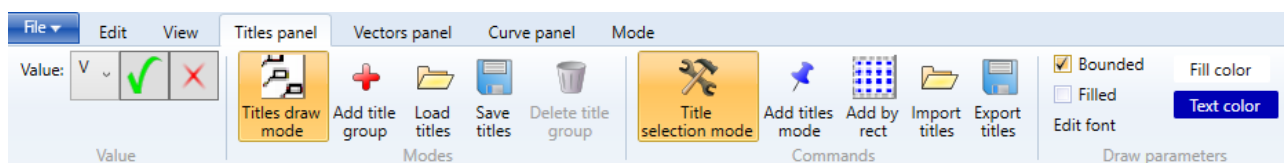


Рисунок 29 – Панель Titles panel в Telma

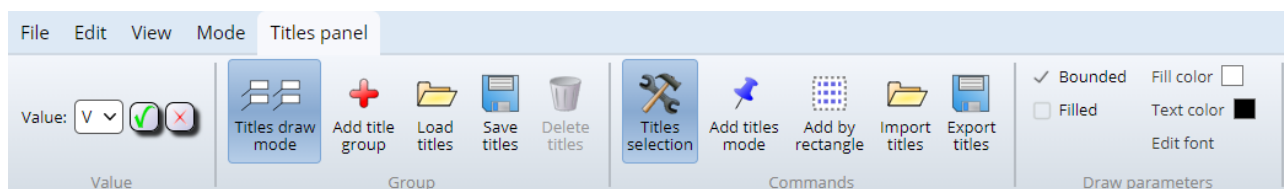


Рисунок 30 – Панель Titles panel в веб-версии

4.3. ПАНЕЛИ В БОКОВОЙ ЧАСТИ ЭКРАНА

На рисунках 31–50 изображены панели в боковой части экрана.



Рисунок 31 – Панель File panel в Telma



Рисунок 32 – Панель File Panel в веб-версии

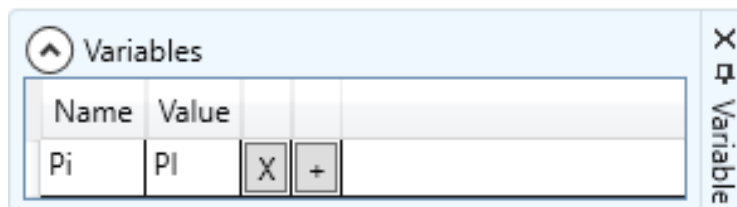


Рисунок 33 – Панель Variables в Telma

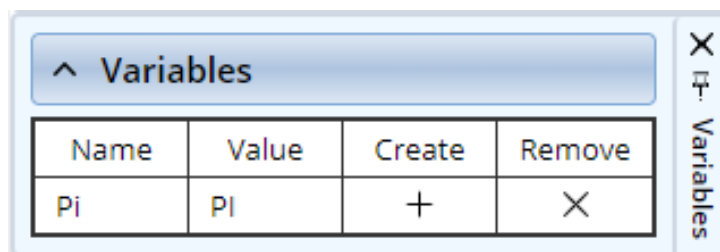


Рисунок 34 – Панель Variables в веб-версии



Рисунок 35 – Панель TelBase main в Telma

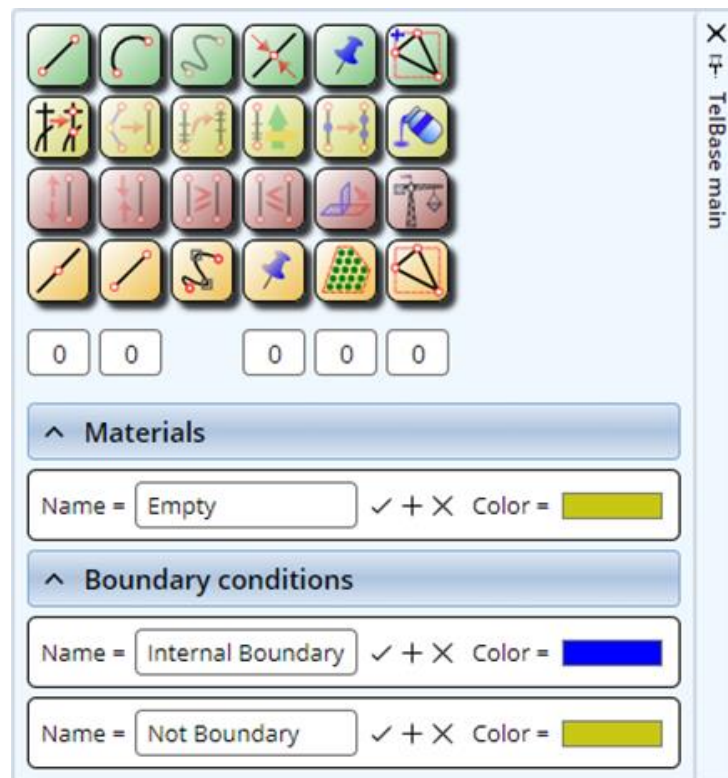


Рисунок 36 – Панель TelBase main в веб-версии

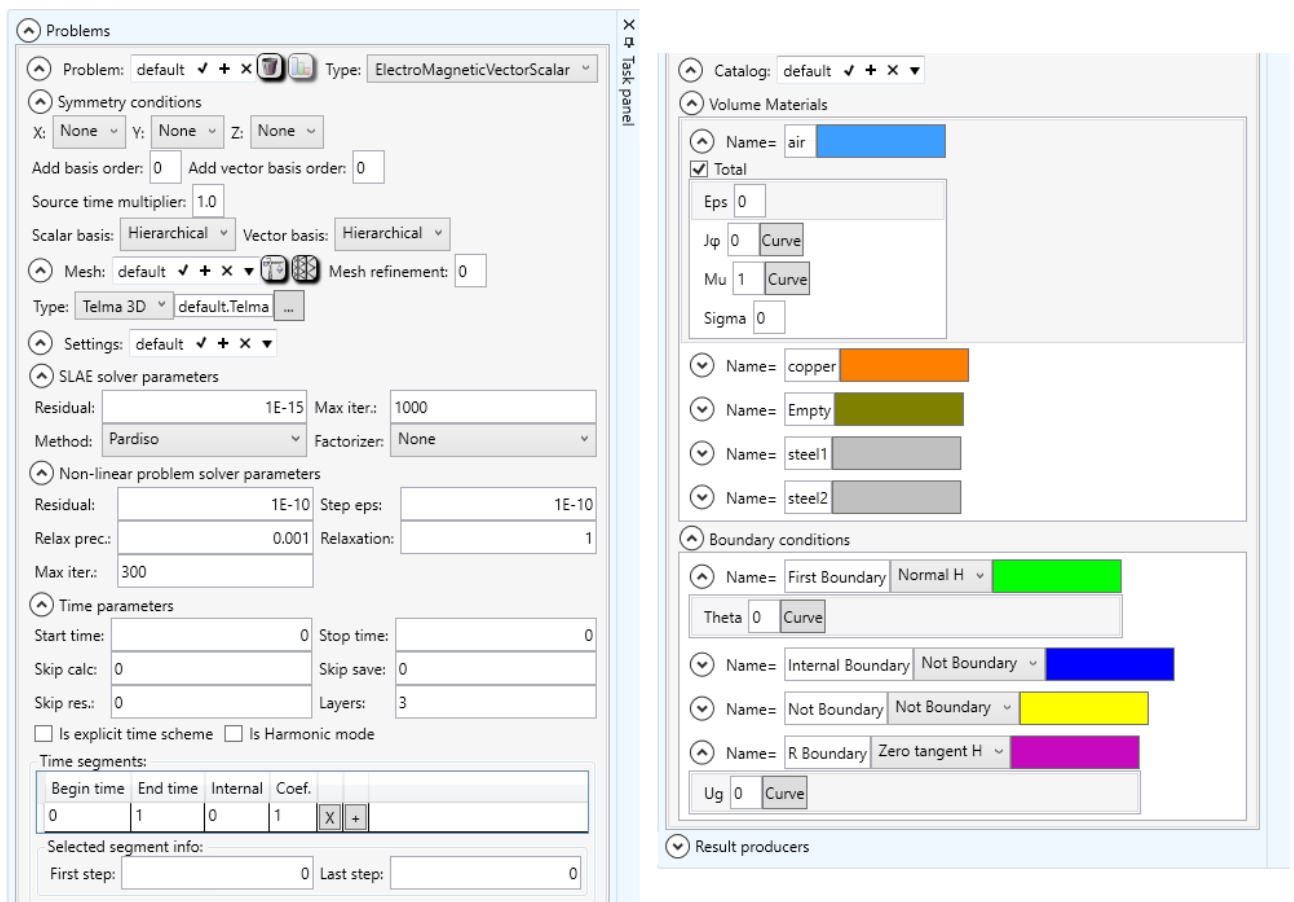


Рисунок 37 – Панель Task panel в Telma



Рисунок 42 – Панель Selection в веб-версии

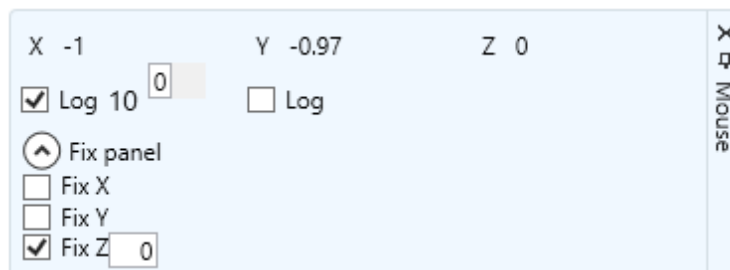


Рисунок 43 – Панель Mouse в Telma

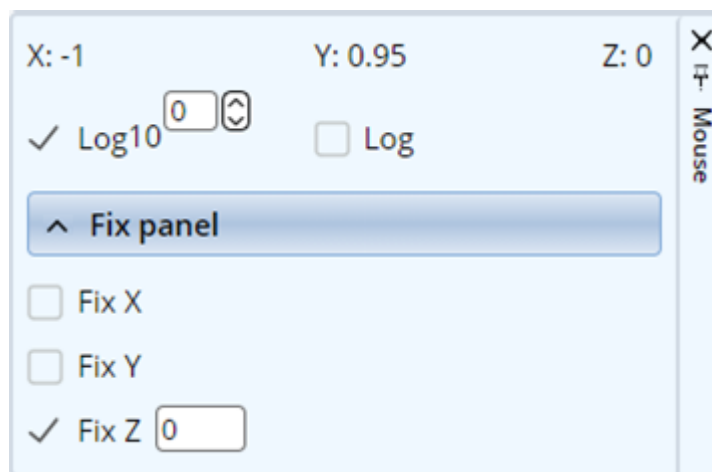


Рисунок 44 – Панель Mouse в веб-версии

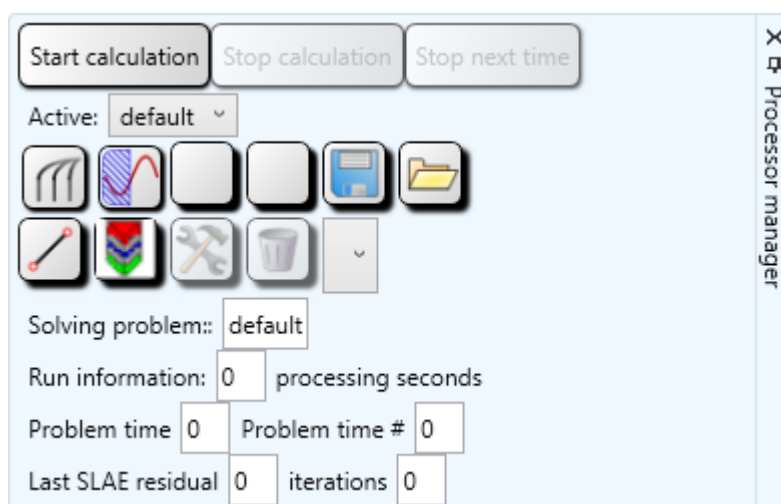


Рисунок 45 – Панель Processor manager в Telma

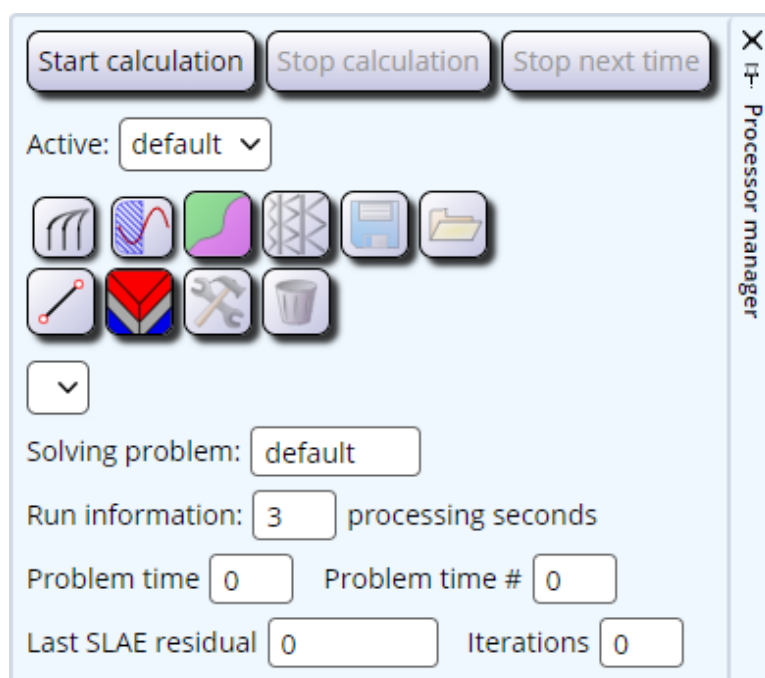


Рисунок 46 – Панель Processor manager в веб-версии

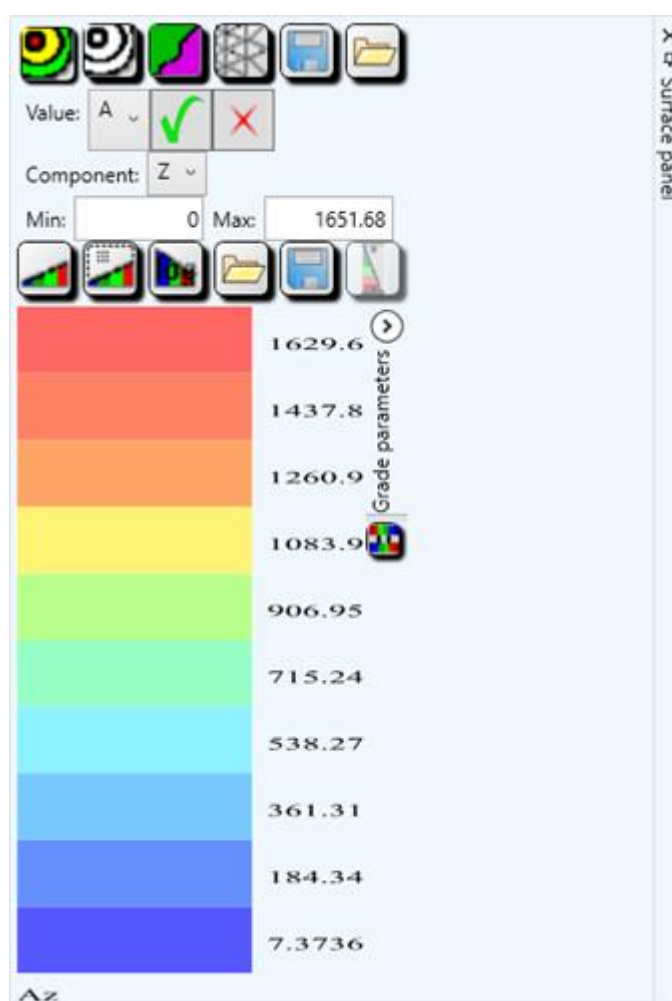


Рисунок 47 – Панель Surface panel в Telma

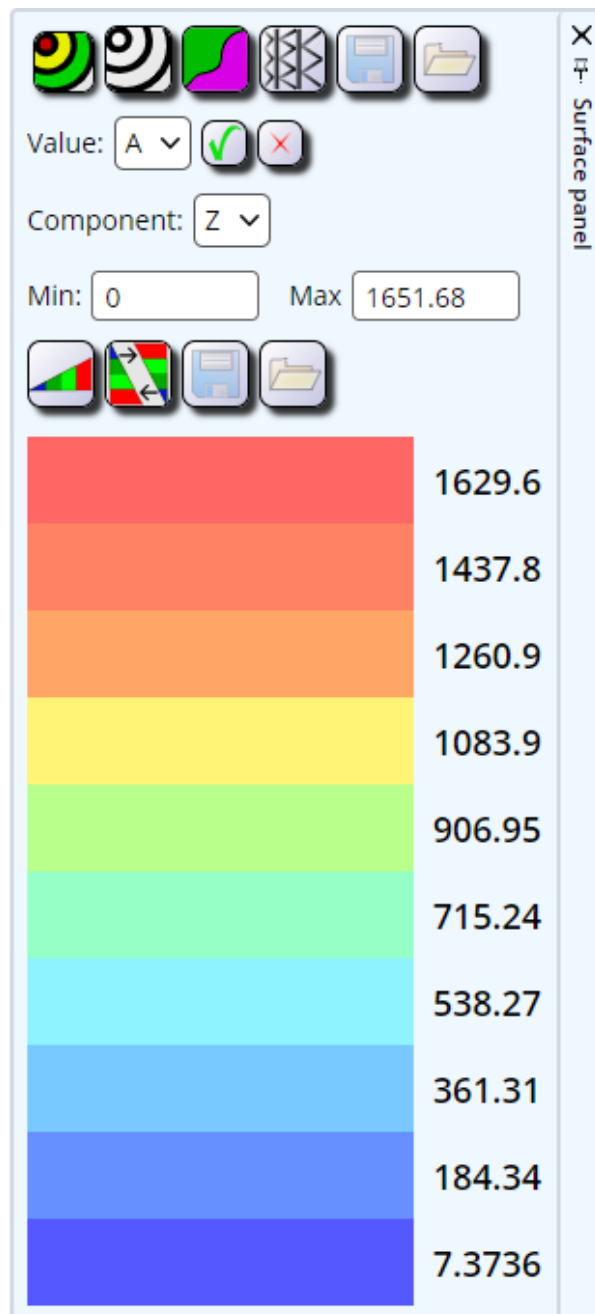


Рисунок 48 – Панель Surface panel в веб-версии

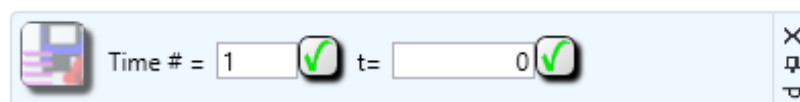


Рисунок 49 – Панель PostProcMainPanel в Telma

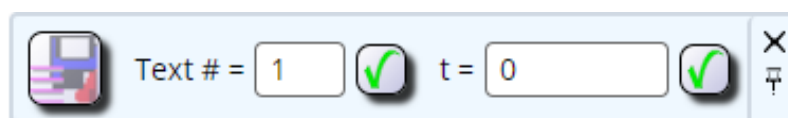


Рисунок 50 – Панель PostProcMainPanel в веб-версии

4.4. ВСПЛЫВАЮЩИЕ ОКНА И СПИСКИ

На рисунках 51–56 изображены всплывающие окна, на рисунках 57–58 представлено меню приложения, а на рисунках 59–64 — всплывающие списки.

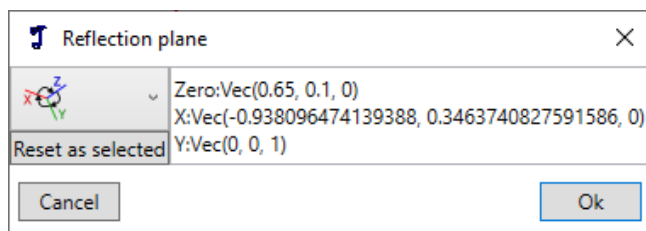


Рисунок 51 – Окно Reflection plane в Telma

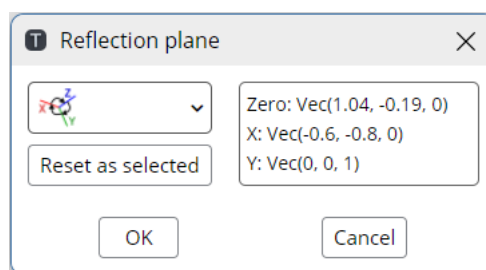


Рисунок 52 – Окно Reflection plane в веб-версии

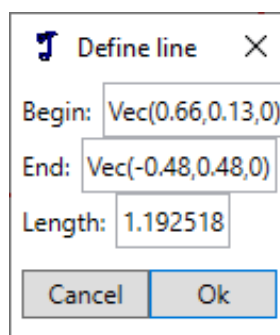


Рисунок 53 – Окно Define line в Telma

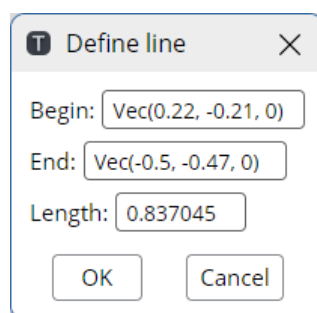


Рисунок 54 – Окно Define line в веб-версии

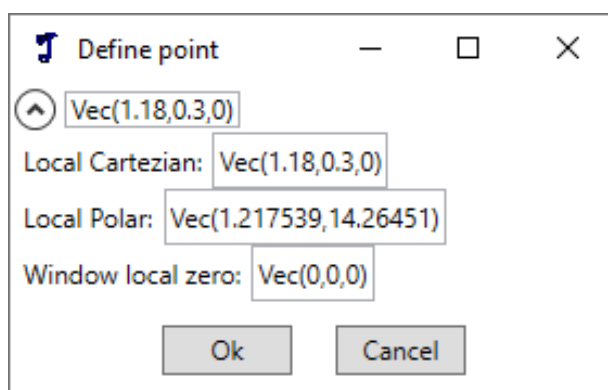


Рисунок 55 – Окно Define point в Telma

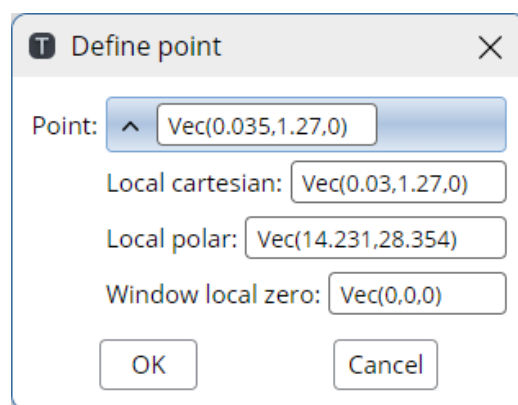


Рисунок 56 – Окно Define point в веб-версии

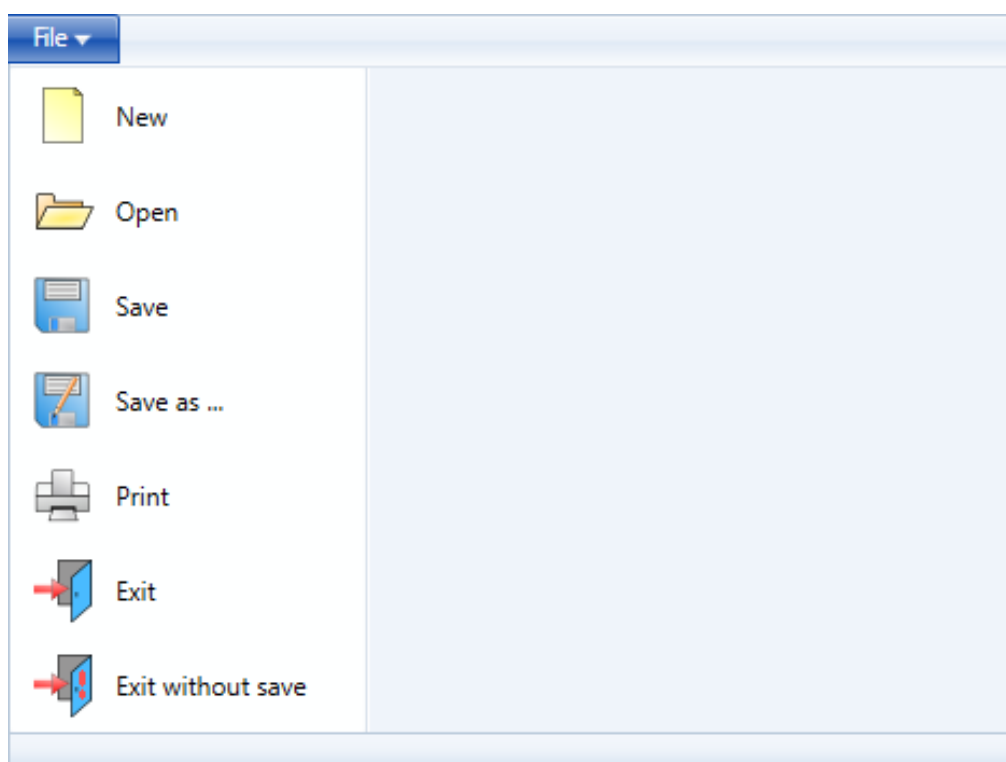


Рисунок 57 – Выпадающее меню приложения в Telma

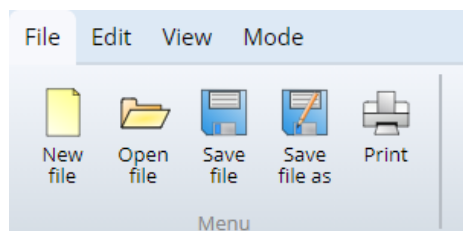


Рисунок 58 – Меню приложения в виде панели в веб-версии

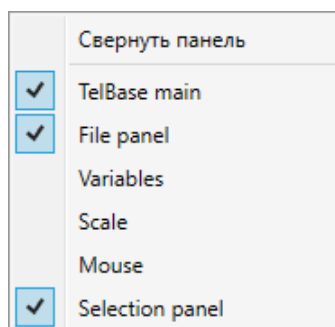


Рисунок 59 – Всплывающий список для управления панелями в Telma

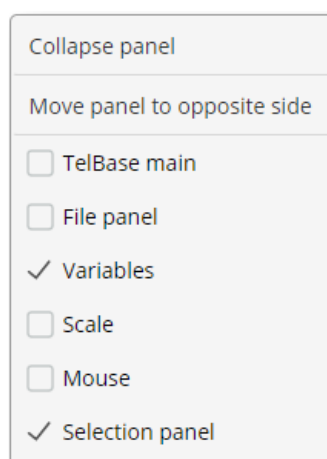


Рисунок 60 – Всплывающий список для управления панелями в веб-версии

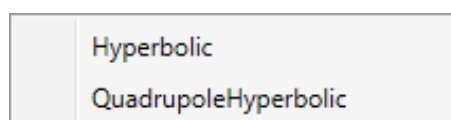


Рисунок 61 – Всплывающий список для выбора сегмента в Telma

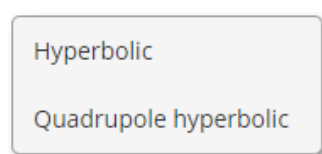


Рисунок 62 – Всплывающий список для выбора сегмента в веб-версии

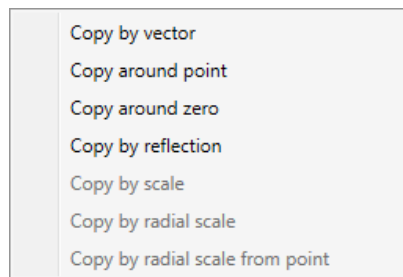


Рисунок 63 – Всплывающий список для выбора типа копирования в Telma

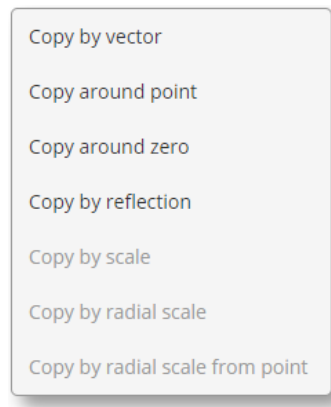




Рисунок 64 – Всплывающий список для выбора типа копирования в веб-версии

4.5. ОКНА ДЛЯ ЗАДАНИЯ ПАРАМЕТРОВ В ПРЕПРОЦЕССОРЕ

На рисунках 65–88 представлены окна для задания параметров объектов.

Рисунок 65 – Окно параметров для прямолинейного сегмента в Telma

Straight segment boundary  

Coefficient =

Internal =

☐ Fix start point:

Local cartesian:

Local polar:

Window local zero:

☐ Fix end point:

Local cartesian:



Local polar:

Window local zero:

Segment length:

Condition:

Рисунок 66 – Окно параметров для прямолинейного сегмента в веб-версии

Arc segment boundary  

Coefficient=

Internal=

☒ Fix start point:

☒ Fix end point:

Segment length:



Condition:

Arc point:

Arc center:

Arc radius:

Рисунок 67 – Окно параметров для криволинейного сегмента в Telma

Arc segment boundary  

Coefficient =

Internal =

☐ Fix start point:

☐ Fix end point:

Segment length:



Condition:

Arc point:


Arc center:

Arc radius:

Рисунок 68 – Окно параметров для криволинейного сегмента в веб-версии

Hyperbolic segment boundary  

Eccentricity


Zero 


Angle

Telma.GUI.ListItem

Coeff

Internal

BegPoint 



EndPoint 

☒ FixedBeg


☐ FixedEnd

Length

Рисунок 69 – Окно параметров для гиперболического сегмента в Telma

Hyperbolic segment boundary  


Eccentricity =


Zero: 

Angle =

Coefficient =



Internal =

☐ Fix start point: 

☐ Fix end point: 


Segment length:


Рисунок 70 – Окно параметров для гиперболического сегмента в веб-версии

Quadrupole Hyperbolic segment boundary  

Coefficient=

Internal=

☒ Fix start point: 

☐ Fix end point: 

Segment length:


Condition: 

Рисунок 71 – Окно параметров для квадрупольного сегмента в Telma

Quadrupole hyperbolic segment boundary ✓ ✕

Coefficient =

Internal =

☐ Fix start point:

☐ Fix end point:

Segment length:

Condition:

Рисунок 72 – Окно параметров для квадрупольного сегмента в веб-версии

Add point on boundary ✓ ✕

PointToAdd

Local Cartesian:

Local Polar:

Window local zero:

Рисунок 73 – Окно параметров для добавления точки в Telma

Add point on boundary ✓ ✕

Point to add:

Local cartesian:

Local polar:

Window local zero:

Рисунок 74 – Окно параметров для добавления точки в веб-версии

Material nail ✓ ✕

Position:

Local Cartesian:

Local Polar:

Window local zero:

Mesh mode:

Element order:

Material:

Рисунок 75 – Окно параметров для добавления материала в Telma

Material nail ☒ ☐

Position:

Local cartesian:

Local polar:

Window local zero:

Mesh mode:

Element order:

Material:

Рисунок 76 – Окно параметров для добавления материала в веб-версии

Node ☒ ☐

Point

Local Carteizian:

Local Polar:

Window local zero:

Рисунок 77 – Окно параметров для выбора узла в Telma

Node ☒ ☐

Point:

Local cartesian:

Local polar:

Window local zero:

Рисунок 78 – Окно параметров для выбора узла в веб-версии

Line ☒ ☐

LastPlaneNormal

FirstPoint

SecondPoint

Length

Рисунок 79 – Окно параметров для перемещения на вектор в Telma

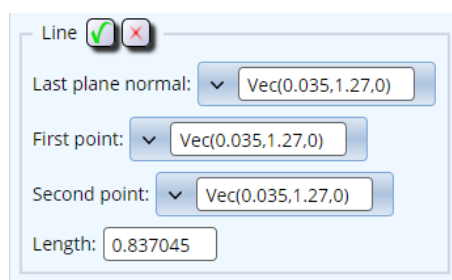


Рисунок 80 – Окно параметров для перемещения на вектор в веб-версии

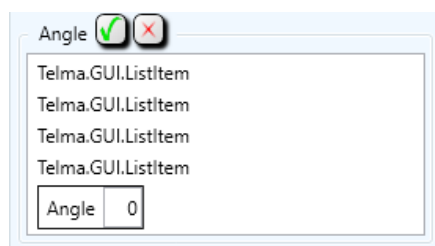


Рисунок 81 – Окно параметров для перемещения вокруг точки в Telma

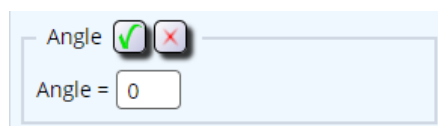


Рисунок 82 – Окно параметров для перемещения вокруг точки в веб-версии

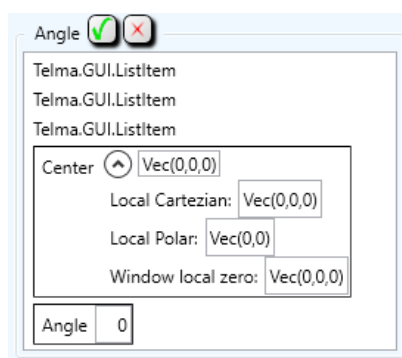


Рисунок 83 – Окно параметров для перемещения вокруг нуля в Telma

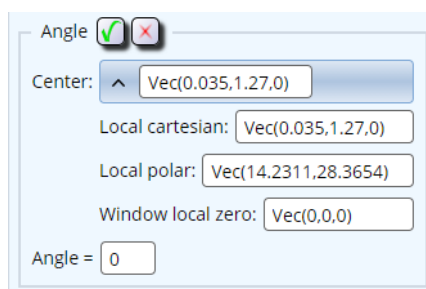


Рисунок 84 – Окно параметров для перемещения вокруг нуля в веб-версии

Рисунок 85 – Окно параметров для выделения прямоугольником в Telma

Рисунок 86 – Окно параметров для выделения прямоугольником в веб-версии

Рисунок 87 – Окно параметров для редактирования границ в Telma

Рисунок 88 – Окно параметров для редактирования границ в веб-версии

4.6. ИНТЕРАКТИВНЫЕ ВОЗМОЖНОСТИ ИНТЕРФЕЙСА

С помощью технологии Blazor и языка JavaScript были реализованы следующие интерактивные возможности интерфейса:

1. Выделение только одной из взаимоисключающих кнопок в некоторых боковых и верхних панелях.
2. Зажатие кнопок при нажатии клавишей мыши.
3. Связь кнопок в боковых формах и верхних панелях для одновременного переключения состояния зажатия.
4. Открепление, закрепление и сворачивание верхних панелей.
5. Открытие и закрытие всплывающих окон и списков.
6. Перемещение боковых панелей и всплывающих окон по экрану.
7. Переключение вкладок для верхних панелей при нажатии на соответствующую вкладку клавишей мыши.
8. Активация чекбоксов и радиокнопок.
9. Открытие и закрытие открывающихся блоков (expander).
10. Валидация значений в полях для работы с числовыми данными.
11. Выделение зафиксированной координаты в нижней части главного окна.
12. Появление выбранной боковой панели над остальными при ее перемещении по экрану.
13. Открепление, закрепление, открытие и закрытие боковых панелей.
14. Связь боковых форм и чекбоксов для них во всплывающем списке для управления панелями.
15. Одновременное сворачивание и разворачивание блока, содержащего боковые панели.
16. Перемещение всех боковых панелей в противоположную сторону экрана.
17. Активация выпадающих списков и выбор одного из значений списка.
18. Изменение системы координат в нижней части препроцессора при выборе соответствующего значения в выпадающем списке.

5. ОПТИМИЗАЦИЯ РАБОТЫ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

5.1. КРИТИЧЕСКИЕ ЭТАПЫ РЕНДЕРИНГА

Рассмотрим некоторые способы оптимизации работы пользовательского веб-интерфейса. Последовательность действий, выполняемых браузером для преобразования HTML, CSS и JavaScript в графическое представление на экране пользователя, называется критическими этапами рендеринга (Critical Rendering Path, CRP). Оптимизация этих действий позволяет повысить производительность и скорость работы рендера веб-страниц, и, как следствие, улучшить пользовательский опыт взаимодействия с интерфейсом. Критические этапы рендеринга включают в себя работу с DOM, CSSOM, деревом рендеринга (Render Tree) и компоновкой объектов документа (Layout).

Чем больше элементов имеет файл с разметкой HTML, тем больше узлов будет содержать DOM-дерево. Поскольку время его построения увеличивается с ростом числа узлов, рекомендуется по возможности уменьшать степень вложенности HTML элементов и сокращать их количество.

На следующем критическом этапе при работе с CSSOM оптимизация не приведет к заметному повышению производительности, поскольку построение CSSOM-дерева в браузере происходит очень быстро. Несмотря на то, что для селекторов с меньшей вложенностью браузеру требуется меньше операций для поиска элемента, сокращение вложенности может привести лишь к незначительному ускорению загрузки страницы.

Для оптимизации этапов работы с деревом рендеринга и компоновкой следует по возможности избегать частого добавления, удаления и модификации узлов DOM-дерева, поскольку это приводит к изменению дерева рендеринга и повторному запуску процесса компоновки. Также для ускорения интерактивного взаимодействия с интерфейсом рекомендуется сокращать количество обновлений экрана.

Во избежание блокировки построения страниц веб-приложения следует также оптимизировать загрузку файлов с JavaScript кодом. Во время загрузки файлов со скриптами браузер не может выполнять обработку событий, таких как прокрутка страницы или нажатия клавиш мыши. Если процессор занят компиляцией и выполнением сценариев JavaScript, то браузер не может реагировать на действия пользователя достаточно быстро. Это приводит к тому, что пользователь видит веб-страницу, но не может взаимодействовать с ней до завершения загрузки, обработки и выполнения JavaScript. В таком случае веб-страницу нельзя назвать интерактивной.

Для оптимизации загрузки файлов с JavaScript кодом был использован сканер предзагрузки. Он занимается обработкой содержимого документа и запрашивает ресурсы с высоким приоритетом, такие как шрифты, CSS и JavaScript. При его использовании не нужно ждать, пока парсер дойдет до фрагмента разметки с вызовом необходимого ресурса. Сканер предзагрузки выполняет запрос нужных данных заранее в фоновом режиме, тем самым уменьшая время блокирования рендера веб-страниц. Благодаря этому, необходимые данные могут загружаться или быть полностью загружены к моменту, когда парсер только начинает обработку фрагмента с их запросом. Для использования сканера предзагрузки необходимо указать у соответствующего HTML тега `<script>` атрибут `defer` или атрибут `async`, если порядок загрузки файлов со скриптами не имеет значения.

5.2. СОКРАЩЕНИЕ РАЗМЕРА ФАЙЛОВ ПРИЛОЖЕНИЯ

Поскольку время загрузки веб-страницы напрямую зависит от размера и количества загружаемых файлов, основные действия по оптимизации должны быть направлены на сокращение объема запрашиваемых ресурсов и количества запросов для их получения. Рассмотрим более подробно некоторые реализованные действия по оптимизации производительности и скорости загрузки созданного интерфейса веб-приложения.

Наиболее эффективным способом сокращения кода является его минификация, то есть процесс удаления из файлов всех символов и фрагментов, не влияющих на работоспособность кода, но увеличивающих его объем. При минификации удалению подлежат лишние пробелы и знаки табуляции, переносы строк, форматирование и все комментарии. Минифицированный код сохраняет работоспособность, но теряет удобочитаемость. По этой причине полная версия файлов обычно сохраняется локально для удобного написания и редактирования кода, а минифицированная версия отправляется в финальную сборку веб-приложения для уменьшения общего объема веб-страниц.

Минификация файлов с разметкой HTML обычно недостаточно эффективна по сравнению с сокращением файлов CSS и JavaScript. Это обусловлено несколькими факторами. Во-первых, файлы HTML чаще всего составляют лишь малую часть от общего объема кода веб-приложений. Во-вторых, вся HTML разметка для динамических сайтов формируется на сервере и отправляется в браузер пользователя при получении запроса. Из этого следует, что минифицированные файлы HTML применяются только на статичных веб-страницах, а реализованное в данной работе веб-приложение является динамическим.

Объем файлов CSS обычно заметно превышает объем файлов HTML. С помощью Gulp все файлы со стилями CSS были объединены с файлом библиотеки `normalize.css`, после чего полученный файл был минифицирован также с использованием Gulp. В результате проделанных действий общий размер всех файлов CSS был уменьшен с 72 Кбайт до 49 Кбайт, то есть примерно в 1.5 раза.

Наибольший объем памяти по сравнению с HTML и CSS занимает код JavaScript, поскольку в веб-приложениях помимо пользовательских скриптов могут также использоваться скрипты различных фреймворков CSS и библиотек JavaScript. Благодаря минификации кода общий размер всех файлов JavaScript в проекте был уменьшен с 292 Кбайт до 91 Кбайт, то есть примерно в 3.2 раза.

Благодаря объединению файлов CSS и JavaScript, количество запросов для получения всех необходимых стилей и скриптов уменьшилось до двух.

Следующим этапом оптимизации была работа с изображениями. Обычно изображения имеют большой объем и, как следствие, загружаются дольше всего. По этой причине важно сократить общий размер всех загружаемых изображений.

В программном комплексе Telma некоторые изображения представлены только в формате PNG, при этом они имеют очень маленький размер в пикселях и недостаточное качество из-за растрового формата. Чтобы иметь возможность использования в интерфейсе любых изображений в любом размере и с высоким качеством, для всех растровых PNG изображений были созданы векторные SVG аналоги. Такая замена изображений позволила сократить их общий объем.

Некоторая часть изображений в Telma помимо представления в формате PNG имеет уже созданный векторный аналог в формате SVG. Имеющиеся векторные изображения имеют устаревший формат и, как следствие, содержат в себе большое количество лишней информации. По этой причине все используемые в интерфейсе SVG изображения, имеющиеся в Telma, были перерисованы в соответствии с новым форматом SVG. Также все созданные изображения были минифицированы для дополнительного сокращения их объема.

В результате описанных действий общий объем всех используемых изображений уменьшился с 1.43 Мбайт до 278 Кбайт, то есть в 5.27 раза.

5.3. ОПТИМИЗАЦИЯ В GOOGLE PAGESPEED INSIGHTS

Поскольку интерфейс был реализован с применением веб-технологий, будем использовать для его тестирования сервис Google PageSpeed Insights. Данный сервис выполняет имитацию загрузки страницы, проводит диагностику производительности и выводит полученные измерения вместе с рекомендациями по улучшению скорости работы веб-приложения. Google PageSpeed Insights также ставит общую оценку производительности тестируемой системы. Оптимальной является оценка в диапазоне 90–100 для мобильной и десктопной версий отдельно. Полный список выполненных действий по оптимизации согласно Google PageSpeed Insights представлен на рисунке 89.

● Устраните ресурсы, блокирующие отображение	▼
● Настройте подходящий размер изображений	▼
● Отложите загрузку скрытых изображений	▼
● Уменьшите размер кода CSS	▼
● Уменьшите размер кода JavaScript	▼
● Удалите неиспользуемый код CSS	▼
● Удалите неиспользуемый код JavaScript	▼
● Настройте эффективную кодировку изображений	▼
● Используйте современные форматы изображений	▼
● Используйте предварительное подключение к необходимым доменам	▼
● Время до получения первого байта от сервера допустимое — Загрузка корневого документа заняла 110 мс	▼
● Избегайте большого количества переадресаций	▼
● Настройте предварительную загрузку ключевых запросов	▼
● Удалите повторяющиеся модули из пакетов JavaScript	▼
● Не отправляйте устаревший код JavaScript в современные браузеры	▼
● Предотвращение чрезмерной нагрузки на сеть — Общий размер достиг 363 КиБ	▼
● Время выполнения кода JavaScript — 0,0 сек.	▼
● Минимизация работы в основном потоке — 0,4 сек.	▼
● Показ всего текста во время загрузки веб-шрифтов	▼
● Уменьшение использования стороннего кода — Сторонний код заблокировал основной поток на 0 мс	▼
● Пассивные прослушиватели событий используются для улучшения производительности при прокрутке	▼
● Метод <code>document.write()</code> не используется	▼
● Избегайте длительных задач в основном потоке	▼

Рисунок 89 – Полный список действий по оптимизации

Оценки производительности мобильной версии веб-приложения до и после оптимизации представлены на рисунках 90 и 91, а оценки для десктопной версии представлены на рисунках 92 и 93 соответственно.

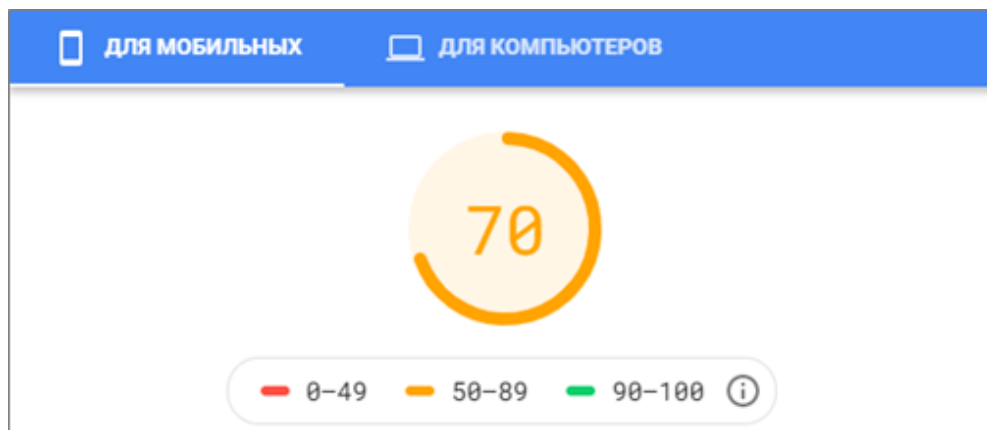


Рисунок 90 – Оценка для мобильной версии до оптимизации

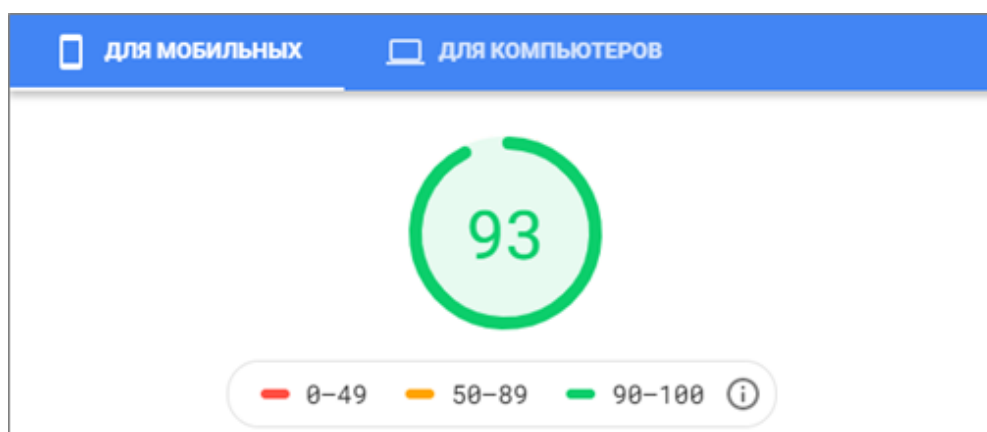


Рисунок 91 – Оценка для мобильной версии после оптимизации

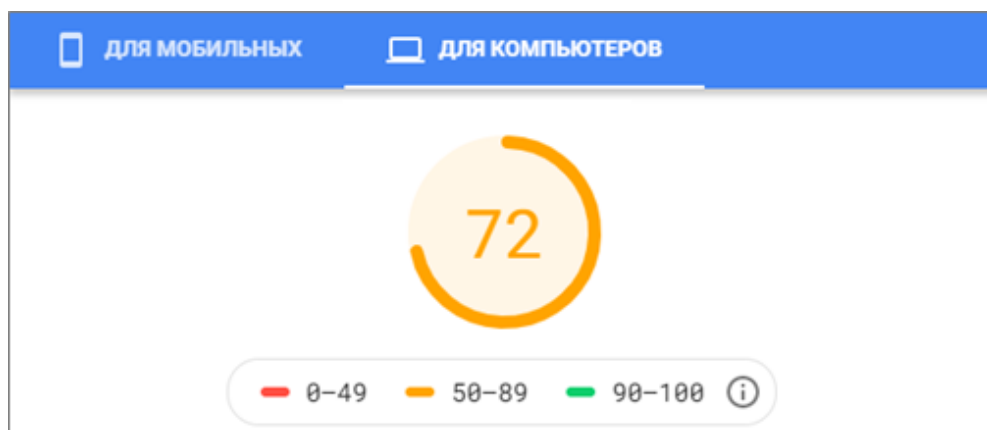


Рисунок 92 – Оценка для десктопной версии до оптимизации

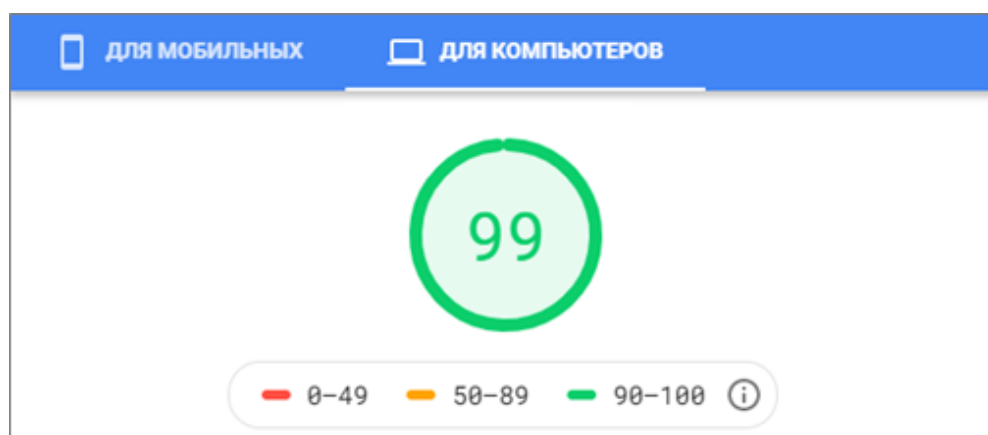


Рисунок 93 – Оценка для десктопной версии после оптимизации

Формирование оценки производится на основе нескольких метрик. Результаты их измерений для страниц до и после оптимизации представлены на рисунках 94 и 95. Пример учета данных метрик при выставлении оценки для десктопной версии приложения после оптимизации представлен на рисунке 96.

<p>● First Contentful Paint 1,1 сек.</p> <p>Первая отрисовка контента – показатель, который определяет интервал времени между началом загрузки страницы и появлением первого изображения или блока текста.</p>	<p>■ Time to Interactive 4,6 сек.</p> <p>Время загрузки для взаимодействия – это время, в течение которого страница становится полностью готова к взаимодействию с пользователем.</p>
<p>● Speed Index 2,6 сек.</p> <p>Индекс скорости загрузки показывает, как быстро на странице появляется контент.</p>	<p>■ Total Blocking Time 400 мс</p> <p>Сумма (в миллисекундах) всех периодов от первой отрисовки контента до загрузки для взаимодействия, когда скорость выполнения задач превышала 50 мс.</p>
<p>■ Largest Contentful Paint 4,0 сек.</p> <p>Отрисовка крупного контента – показатель, который определяет время, требуемое на полную отрисовку крупного текста или изображения.</p>	<p>● Cumulative Layout Shift 0</p> <p>Совокупное смещение макета – это процентная величина, на которую смещаются видимые элементы области просмотра при загрузке.</p>

Рисунок 94 – Результаты имитации загрузки страницы до оптимизации

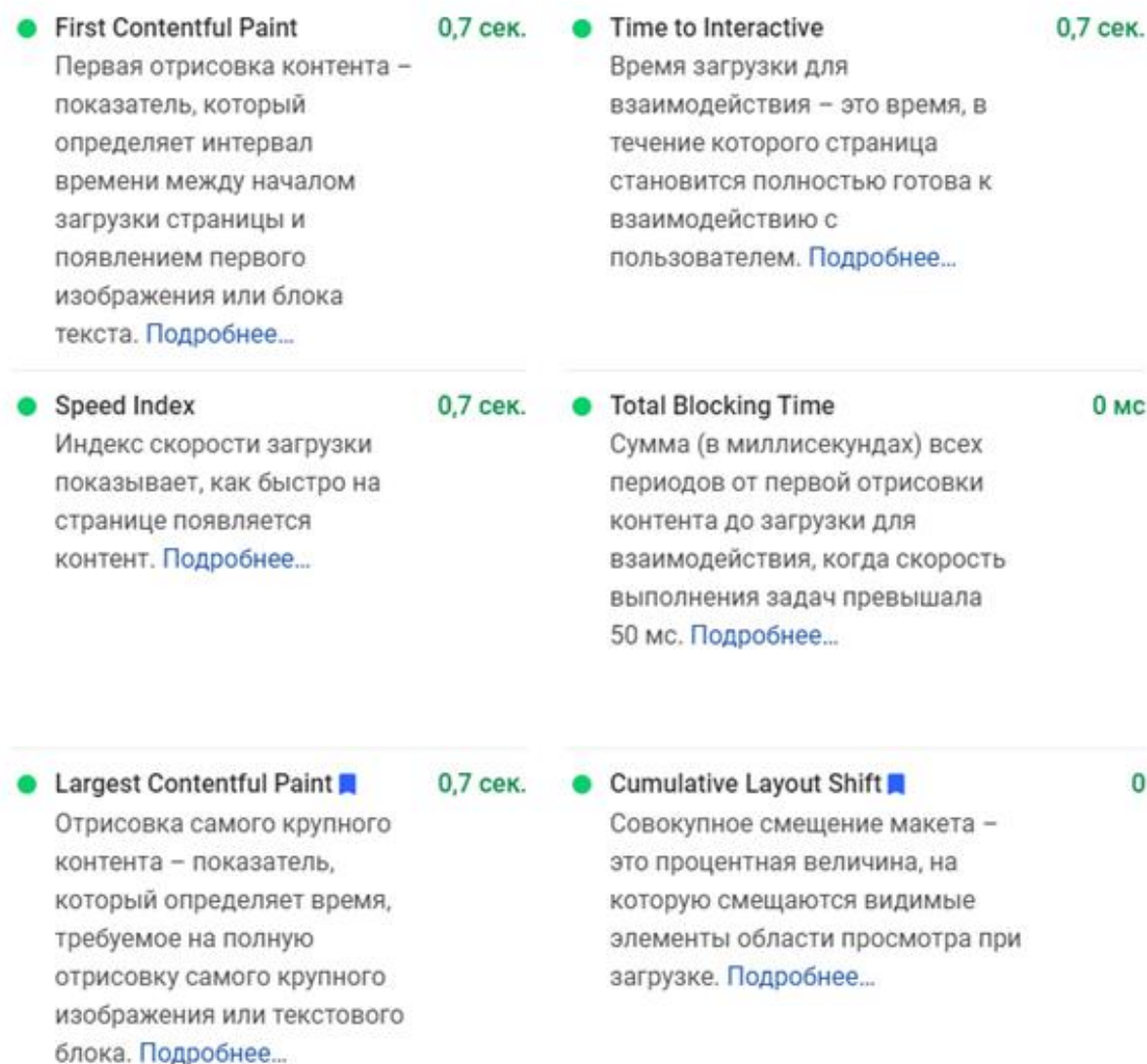


Рисунок 95 – Результаты имитации загрузки страницы после оптимизации

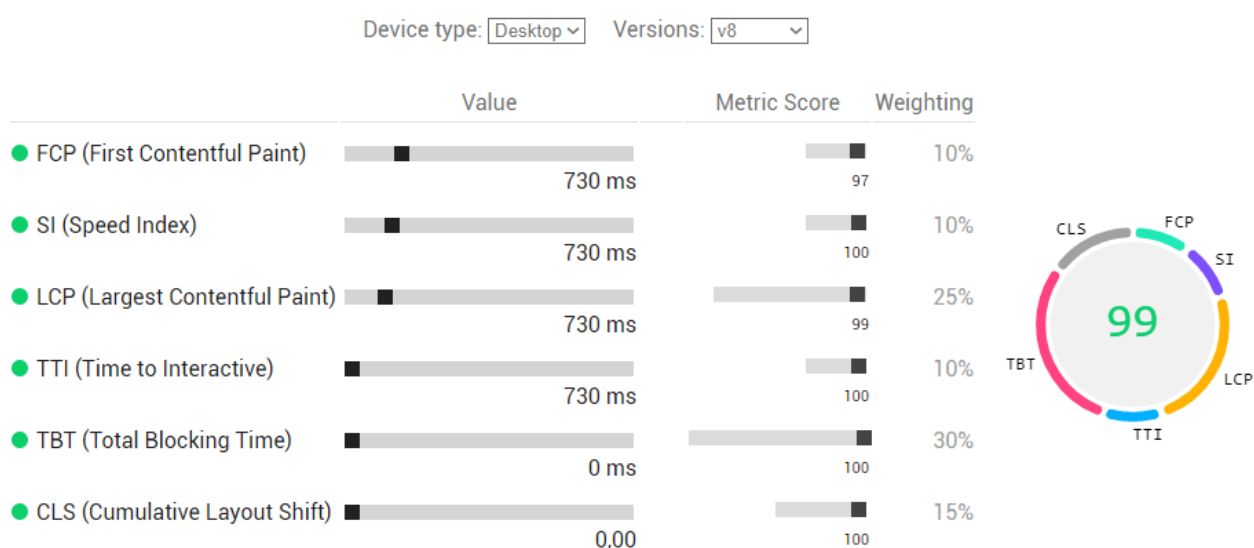


Рисунок 96 – Пример набора метрик для расчета оценки веб-приложения

6. ТЕСТИРОВАНИЕ VIEW С ПОМОЩЬЮ РАСЧЕТА КОНЕЧНОЭЛЕМЕНТНОЙ ЗАДАЧИ

Протестируем реализованный функционал MVVM View для двумерной части программного комплекса Telma. Тестирование программы будем производить с помощью расчёта конечноэлементной задачи с последующим сравнением результатов решения в Telma и приложении Blazor. В качестве образца для сравнения выберем электромагнитную задачу, предоставленную сотрудниками Института ядерной физики им. Г. И. Будкера СО РАН. Данная задача более подробно описана в работах Авдеевой С.Д., Громовой Т.В. и Дербышевой Т.Р.

Модель для расчета задачи представлена на рисунке 97.

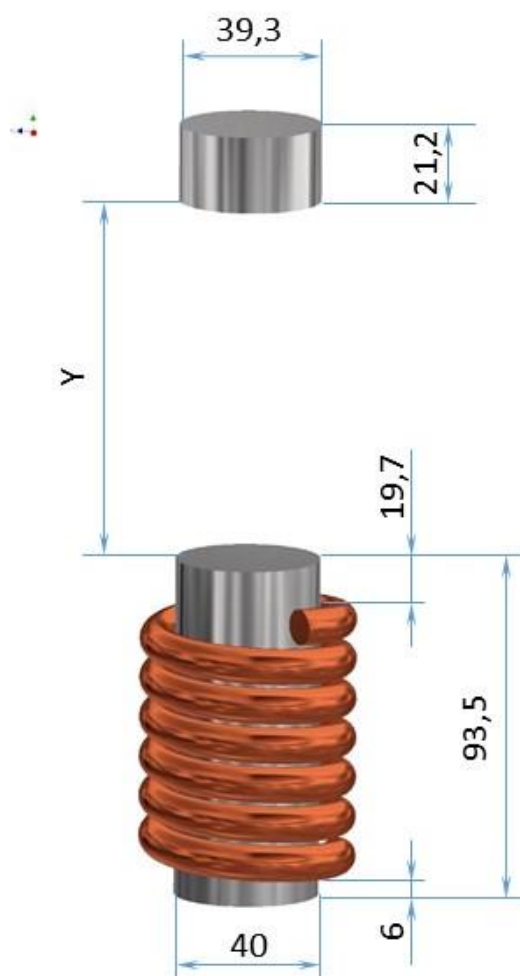


Рисунок 97 – Модель для расчета конечноэлементной задачи

Модель для расчета представляет собой два стальных цилиндра, находящихся на некотором заданном расстоянии друг от друга. Верхний цилиндр имеет высоту 21.2 мм и диаметр 39.3 мм, а нижний — высоту 93.5 мм и диаметр 40 мм. На нижнем цилиндре находится катушка, состоящая из 291 витка. Диаметр провода катушки составляет 0.8 мм. Расстояние от верхнего конца цилиндра до обмотки составляет 19.7 мм, а от нижнего конца до обмотки — 6 мм.

Расчетная область была задана в двумерных цилиндрических координатах при расстоянии 0.005 м между верхним и нижним цилиндрами. Область образована сечением $\varphi = \text{const}$ и включает в себя бак, внутри которого располагаются два цилиндра и область сторонного тока. Бак задан в виде прямоугольной области $\Omega_1 = [10^{-6}, 1] \times [-0.44015, 0.55985]$, нижний цилиндр — в виде прямоугольной области $\Omega_2 = [10^{-6}, 0.02] \times [0, 0.0935]$, верхний цилиндр — в виде прямоугольной области $\Omega_3 = [10^{-6}, 0.01965] \times [0.0985, 0.1197]$, область сторонного тока — в виде прямоугольной области $\Omega_4 = [0.022, 0.0252] \times [0.006, 0.0738]$. Все указанные размеры представлены в системе СИ. Бак состоит из воздуха с характеристиками $J_\varphi = 0$, $\mu = 1$, верхний и нижний цилиндры — из магнитной стали с характеристиками $J_\varphi = 0$, $\mu = 1000$, а область сторонного тока имеет характеристики

$$J_\varphi = \frac{\text{сила тока} \cdot \text{количество витков}}{\text{площадь поперечного сечения}} = \frac{5 \cdot 291}{0.0032 \cdot 0.0678}, \mu = 1.$$

Поскольку требуется проверить корректность работы реализованных возможностей веб-интерфейса для препроцессора, процессора и постпроцессора, выполним задание описанной ранее задачи в приложении Blazor с самого начала в соответствии с образцом. Для этого в препроцессоре построим конечноэлементную сетку и добавим материалы и краевые условия, в процессоре зададим параметры задачи и запустим её решение, а в постпроцессоре выведем полученное распределение поля вектор-потенциала [4]. Результаты решения линейной задачи в сравнении с образцом представлены на рисунках 98 и 99 соответственно.

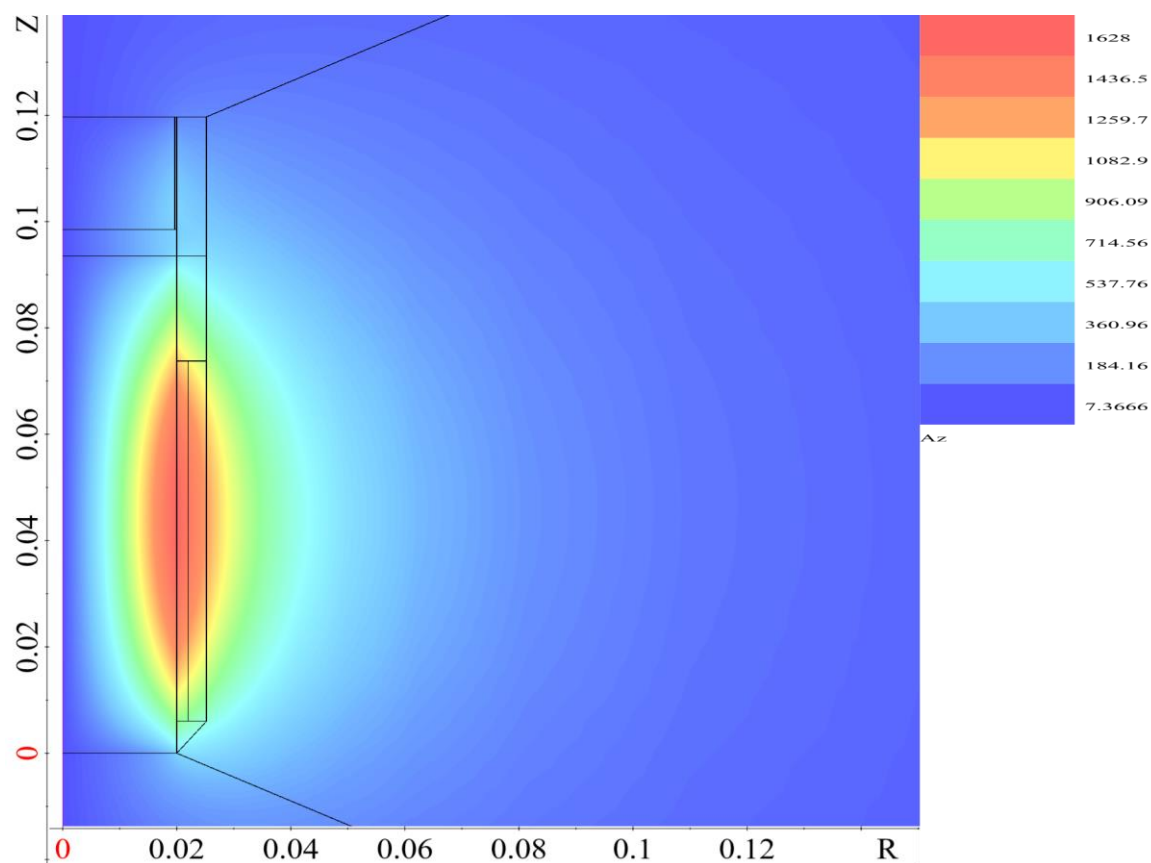


Рисунок 98 – Распределение поля вектор-потенциала в Telma

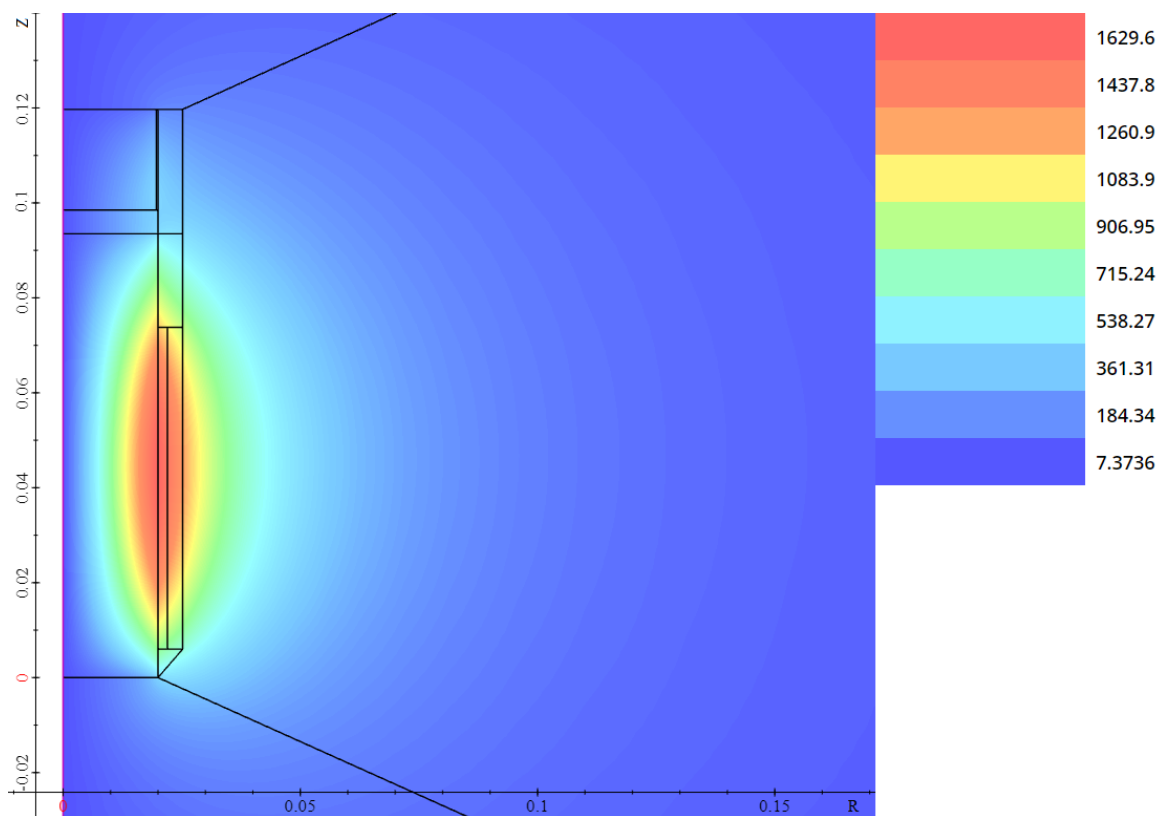


Рисунок 99 – Распределение поля вектор-потенциала в веб-версии

ЗАКЛЮЧЕНИЕ

1. Рассмотрены принципы работы веб-технологий HTML, CSS, JavaScript и Blazor и их использование для создания пользовательских веб-интерфейсов. На основе данных технологий был реализован полноценный кроссплатформенный веб-интерфейс двумерной части программного комплекса Telma.

2. Для элементов управления Telma, созданных с помощью WPF и XAML, в веб-интерфейсе были реализованы соответствующие аналоги. Для всех компонентов создана подробная документация с описанием списка возможных атрибутов и их допустимых значений.

3. Дополнительно рассмотрены способы оптимизации процесса разработки веб-приложения и его производительности. Процесс разработки может быть автоматизирован и значительно упрощен благодаря использованию различных библиотек, CSS-препроцессоров и сборщиков проектов. Скорость работы веб-приложения может быть оптимизирована в основном с помощью сжатия файлов, использования современных форматов изображений и сокращения количества ресурсов и запросов для их получения.

4. В результате проведенного исследования можно сделать вывод, что MVVM View на основе WPF может быть перенесен на технологию Blazor благодаря возможности использования компонентного подхода при создании элементов интерфейса. Тем не менее перенос интерфейса не может быть автоматизирован с помощью создания транслятора, поскольку не для всех элементов управления WPF существуют готовые аналоги в HTML, а их самостоятельное создание является достаточно трудоемким.

СПИСОК ЛИТЕРАТУРЫ

1. Баранов А. В. Проектная разработка виртуальной лабораторной работы с 3D-визуализацией движения гироскопа / А. В. Баранов, И. Д. Мурамщиков, Н. А. Скрынник // Электронные средства и системы управления = Electronic Devices and Control Systems: материалы докл. 15 междунар. науч.-практ. конф., Томск, 20–22 ноября 2019 г.: в 2 ч. – Томск: В-Спектр, 2019. – ч. 2. – с. 170–172. – 100 экз. – ISBN 978-5-91191-428-8.
2. Лоусон Б. Изучаем HTML5. Библиотека специалиста. 2-е издание. / Лоусон Б., Шарп Р. – СПб: Питер, 2012. – 304 с. – ISBN 978-5-459-01156-2.
3. Мурамщиков И. Д. Исследование эффективности использования электронных учебных пособий, созданных с применением WEB-технологий / И. Д. Мурамщиков, Н. А. Скрынник, С. Х. Рояк // Наука. Технологии. Инновации: сб. науч. тр.: в 9 ч., Новосибирск, 30 ноября – 4 декабря 2020 г. – Новосибирск: Изд-во НГТУ, 2020. – ч. 8. – с. 335–340. – 100 экз. – ISBN 978-5-7782-4296-8.
4. Соловейчик Ю.Г. Метод конечных элементов для решения скалярных и векторных задач: учеб. пособие / Ю.Г. Соловейчик, М.Э. Рояк, М.Г. Персова – Новосибирск: Изд-во НГТУ, 2007. – 896 с. – ISBN 978-5-7782-0749-9.
5. Флэнаган Д. JavaScript. Подробное руководство, 6-е издание. – Пер. с англ. – СПб: Символ-Плюс, 2012. – 1080 с. – ISBN: 978-5-93286-215-5.
6. Шмитт К. CSS. Рецепты программирования. – СПб.: BHV, 2011. – 672 с. – ISBN 978-5-9775-0649-6.
7. Baranov A. V. Project development of the virtual laboratory work with 3D visualization of gyroscope motion / A. V. Baranov, I. D. Muramshchikov, N. A. Skrynnik // Journal of Physics: Conference Series. – 2020. – Vol. 1488: International Scientific Conference on Electronic Devices and Control Systems (EDCS 2019), Tomsk, 2019. – Art. 012005 (7 p.). – DOI: 10.1088/1742-6596/1488/1/012005.
8. Booch G. Object-Oriented Analysis and Design with Applications – Third Edition. / Booch G., Maksimchuk R.A., Engel M.W., Young B.J., Conallen J. – Boston: Addison-Wesley Professional, 2007. – 720 p. – ISBN 978-0-2018-9551-3.

9. Fenton S. Pro TypeScript: Application-Scale JavaScript Development. – First Edition. – Germany: Apress, 2014. – 248 p. – ISBN 978-1-4302-6791-1.
10. Himschoot P. Microsoft Blazor: Building Web Applications in .NET. – Germany: Apress, 2020. – 277 p. – ISBN 978-1-4842-5927-6.
11. MacDonald M. Pro WPF 4.5 in C#: Windows Presentation Foundation in .NET 4.5. – Germany: Apress, 2012. – 1112 p. – ISBN 978-1-4302-4365-6.
12. Nathan A. XAML Unleashed – First Edition. – USA: SAMS, 2014. – 498 p. – ISBN 978-0-6723-3722-2.
13. Price M.J. C# 9 and .NET 5 – Modern Cross-Platform Development: Build intelligent apps, websites, and services with Blazor, ASP.NET Core, and Entity Framework Core using Visual Studio Code – Fifth Edition. – Birmingham: Packt Publishing, 2020. – 822 p. – ISBN 978-1-8005-6810-5.
14. Yuen S. Mastering Windows Presentation Foundation: Build responsive UIs for desktop applications with WPF – Second Edition. – Birmingham: Packt Publishing, 2020. – 636 p. – ISBN 978-1-8386-4341-6.

ПРИЛОЖЕНИЕ А. ТЕКСТ ПРОГРАММЫ

ДЛЯ ЦЕЛОЧИСЛЕННОГО ПОЛЯ INTINPUT

```
@switch (InputType)
{
    case InputType.Table:
        <input class="table__input @(IsVisible ? "" : "hide")"
            style=@(ValidationError ? "border: 1px solid red;" : "")
            type="number"
            onPaste="return false"
            readonly=@IsReadonly
            value=@Source.ToString()
            @onchange=@OnChangeImpl>
        break;

    case InputType.Default:
        <Wrapper IsVisible=@IsVisible>
            <Label Text=@Text />
            <input class="input @SizeTypeMap[SizeType]"
                style=@(ValidationError ? "border: 1px solid red;" : "")
                type="number"
                onPaste="return false"
                readonly=@IsReadonly
                value=@Source.ToString()
                @onchange=@OnChangeImpl
                @onclick:stopPropagation="true">
        </Wrapper>
        break;

    default:
        break;
}

@code
{
    [Parameter] public string Text { get; set; } = "";

    [Parameter] public int Source { get; set; } = 0;

    [Parameter] public int Min { get; set; } = -10000;
    [Parameter] public int Max { get; set; } = 10000;

    [Parameter] public Action<int> OnChange { get; set; } = null;

    [Parameter] public InputSize SizeType {get; set;} = InputSize.Stretch;
    [Parameter] public InputType InputType { get; set; } =
        InputType.Default;

    [Parameter] public bool IsReadonly { get; set; } = false;

    [Parameter] public bool IsVisible { get; set; } = true;

    bool ValidationError = false;
```

```

Dictionary<InputSize, string> SizeTypeMap =
new Dictionary<InputSize, string>()
{
    [InputSize.Short] = "input-short",
    [InputSize.Middle] = "input-middle",
    [InputSize.Long] = "input-long",
    [InputSize.Stretch] = ""
};

void OnChangeImpl(ChangeEventArgs args)
{
    if (int.TryParse((string)args.Value, out int result))
    {
        if (result >= Min && result <= Max)
        {
            Source = result;
            ValidationError = false;

            OnChange?.Invoke(Source);
        }
        else
            ValidationError = true;
    }
    else
        ValidationError = true;
}
}

```

ПРИЛОЖЕНИЕ Б. ТЕКСТ ПРОГРАММЫ ДЛЯ DICTIONARYSELECT

```
<div class="select-box @OrientationMap[Orientation] @SideMap[Side]"
    title=@Title>
    <div class="options-container @(IsActive ? "active" : "")"
        @onclick=@(() => IsActive = !IsActive) @onclick:stopPropagation="true">
        @foreach (var pair in Source)
        {
            <div class="option" @onclick=@(() => OnChangeImpl(pair.Key))>
                <input type="radio" class="radio" />
                <label>@pair.Value</label>
            </div>
        }
    </div>

    <div class="selected" @onclick=@(() => IsActive = !IsActive)
        @onclick:stopPropagation="true">
        @Source[Selected]
    </div>
</div>

@code
{
    [Parameter] public Dictionary<string, string> Source { get; set; }
    [Parameter] public Action<string> OnChange { get; set; } = null;
    [Parameter] public string Selected { get; set; } = "";
    [Parameter] public string Title { get; set; } = "";
    [Parameter] public Orientation Orientation { get; set; } =
                                                Orientation.Bottom;
    [Parameter] public Side Side { get; set; } = Side.Left;

    Dictionary<Orientation, string> OrientationMap =
    new Dictionary<Orientation, string>()
    {
        [Orientation.Top] = "select-top",
        [Orientation.Bottom] = ""
    };

    Dictionary<Side, string> SideMap = new Dictionary<Side, string>()
    {
        [Side.Right] = "select-right",
        [Side.Left] = ""
    };

    bool IsActive { get; set; } = false;

    void OnChangeImpl(string value)
    {
        Selected = value;
        OnChange?.Invoke(value);
    }
}
```

ПРИЛОЖЕНИЕ В. ТЕКСТ ПРОГРАММЫ ДЛЯ ЭЛЕМЕНТА ВЫБОРА ЧЕКСБОВ

```

<div class="checkbox-container @(IsEnabled ? "" : "disabled")
      @(IsVisible ? "" : "hide")">
  <input type="checkbox" class="input-checkbox" id=@Id
        @onChange=@OnChangeImpl checked=@Source />
  <label for=@Id data-text=@Text
        class="checkbox @CheckboxSizeMap[CheckboxSize]
              @CheckboxTypeMap[CheckboxType]">
    <span class="check">
      <svg width="18px" height="18px" viewBox="0 0 18 18">
        <path d="M1,9 L1,3.5 C1,2 2,1 3.5,1 L14.5,1 C16,1 17,2
              17,3.5 L17,14.5 C17,16 16,17 14.5,17 L3.5,17
              C2,17 1,16 1,14.5 L1,9 Z">
        </path>
        <polyline points="0 8 6 13 14 3"></polyline>
      </svg>
    </span>
    @Text
  </label>

  @ChildContent
</div>

@code
{
  [Parameter] public bool Source { get; set; } = false;
  [Parameter] public RenderFragment ChildContent { get; set; } = null;
  [Parameter] public string Text { get; set; } = "";
  [Parameter] public string Id { get; set; } = "";
  [Parameter] public Action<bool> OnChange { get; set; } = null;
  [Parameter] public bool IsEnabled { get; set; } = true;
  [Parameter] public bool IsVisible { get; set; } = true;

  [Parameter] public CheckboxType CheckboxType { get; set; } =
                                                    CheckboxType.Default;
  Dictionary<CheckboxType, string> CheckboxTypeMap =
  new Dictionary<CheckboxType, string>() {
    [CheckboxType.Input] = "with_input",
    [CheckboxType.Default] = ""
  };

  [Parameter] public CheckboxSize CheckboxSize { get; set; } =
                                                    CheckboxSize.Default;
  Dictionary<CheckboxSize, string> CheckboxSizeMap =
  new Dictionary<CheckboxSize, string>() {
    [CheckboxSize.Small] = "checkbox-small",
    [CheckboxSize.Default] = ""
  };

  void OnChangeImpl(ChangeEventArgs args) =>
    OnChange?.Invoke((bool) args.Value);
}

```

ПРИЛОЖЕНИЕ Г. ТЕКСТ ПРОГРАММЫ ДЛЯ ЭЛЕМЕНТА ВЫВОДА СПИСКА LIST

```
<div class="select-box @(IsVisible ? "" : "hide")">
  @if (Source != null)
  {
    @foreach (var value in Source)
    {
      <div class="option @(Selected == value ? "active" : "")"
        data-name=@value
        @onclick=@(() => OnChangeImpl(value))>
        <input type="radio"
          class="radio" />
        <label>
          @value
        </label>
      </div>
    }
  }
</div>

<Label Text=@(Selected != "" && IsVisible ? Text + Selected : "") />

@code
{
  [Parameter] public IEnumerable<string> Source { get; set; } = null;
  [Parameter] public Action<string> OnChange { get; set; } = null;

  [Parameter] public string Selected { get; set; } = "";
  [Parameter] public string Text { get; set; } = "";

  [Parameter] public bool IsVisible { get; set; } = true;

  void OnChangeImpl(string value)
  {
    Selected = value;
    OnChange?.Invoke(value);
  }
}
```


ПРИЛОЖЕНИЕ Д. ТЕКСТ ПРОГРАММЫ ДЛЯ БЛОКА EXPANDER

```
<div class="expander @ExpanderTypeMap[ExpanderType]
      @(IsVisible ? "" : "hide")">
  <div class="expander__block block">
    <div class="block__heading @(IsExpanded ? "active" : "")"
      @onclick=@(() => IsExpanded = !IsExpanded)>
      @if (ExpanderType == ExpanderType.Horizontal)
      {
        @Heading
      }
      else
      {
        
        <span>
          @Text
        </span>
      }
    </div>

    <div class="block__collapse @(IsExpanded ? "active" : "")">
      @Content
    </div>
  </div>
</div>

@code
{
  [Parameter] public string Text { get; set; } = "";

  [Parameter] public RenderFragment Heading { get; set; } = null;
  [Parameter] public RenderFragment Content { get; set; } = null;

  [Parameter] public bool IsVisible { get; set; } = true;
  [Parameter] public bool IsExpanded { get; set; } = false;

  [Parameter] public ExpanderType ExpanderType { get; set; } =
                                          ExpanderType.Horizontal;

  Dictionary<ExpanderType, string> ExpanderTypeMap =
  new Dictionary<ExpanderType, string>()
  {
    [ExpanderType.Vertical] = "expander-vertical",
    [ExpanderType.Horizontal] = ""
  };
}
```

ПРИЛОЖЕНИЕ Е. ТЕКСТ ПРОГРАММЫ ДЛЯ КНОПКИ С КАРТИНКОЙ IMAGEBUTTON

```
@switch (ButtonType)
{
    case ImageButtonType.Icon:
        <button class="button
            button-icon
            @Class
            @(IsVisible ? "" : "hide")
            @(OnClickEvent.HasDelegate ? "" : "disabled")"
            title=@Title
            @onclick=@OnClickEvent
            @onclick:stopPropagation="true">
            <img src=@Image alt=@Title />
        </button>
        break;

    case ImageButtonType.Default:
        <button class="button
            button-image
            @ButtonSizeMap[ButtonSize]
            @(CanPush && IsPushed ? "push" : "")
            @(Disabled ? "disabled" : "")
            @(IsVisible ? "" : "hide")"
            title=@Title
            @onclick=@(() => Command?.Execute(null))
            @onclick:stopPropagation="true">
            <img src=@Image alt=@Title />
        </button>
        break;

    default:
        break;
}

@code
{
    [Parameter] public string Image { get; set; } = "";
    [Parameter] public string Title { get; set; } = "";

    [Parameter] public string Class { get; set; } = "";

    [Parameter] public EventCallback OnClickEvent { get; set; } = null;
    [Parameter] public ITelmaCommand Command { get; set; } = null;

    [Parameter] public ImageButtonType ButtonType { get; set; } =
        ImageButtonType.Default;

    [Parameter] public bool CanPush { get; set; } = false;
    [Parameter] public bool IsPushed { get; set; } = false;

    [Parameter] public bool IsVisible { get; set; } = true;
```

```

[Parameter] public ButtonSize ButtonSize {get; set;} = ButtonSize.Big;

Dictionary<ButtonSize, string> ButtonSizeMap =
new Dictionary<ButtonSize, string>() {
    [ButtonSize.Tiny] = "button-image-tiny",
    [ButtonSize.Small] = "button-image-small",
    [ButtonSize.Middle] = "button-image-middle",
    [ButtonSize.Big] = ""
};

bool Disabled => Command != null ? !Command.CanExecute(null) : true;

protected override void OnParametersSet()
{
    base.OnParametersSet();

    if (Command != null)
        Command.CanExecuteChanged += (sender, args) =>
            InvokeAsync(() => StateHasChanged());
}
}

```

ПРИЛОЖЕНИЕ Ж. ТЕКСТ ПРОГРАММЫ ДЛЯ КНОПКИ С ТЕКСТОМ TEXTBUTTON

```
@switch (ButtonType)
{
    case TextButtonType.Outline:
        <button class="button button-outline
            @(CanPush && IsPushed ? "push" : "")
            @(Disabled ? "disabled" : "")
            @(IsVisible ? "" : "hide")"
            title=@Title @onclick=@OnClick
            @onclick:stopPropagation="true">
            @Text
        </button>
        break;

    case TextButtonType.Color:
        <label class="button button-text button-color
            @(Disabled ? "disabled" : "")
            @(IsVisible ? "" : "hide")"
            for=@Id
            title=@Title
            @onclick=@OnClick
            @onclick:stopPropagation="true">
            <span>@Text</span>
            <input type="color" id=@Id value=@Source>
        </label>
        break;

    case TextButtonType.Popup:
        <button class="button button-text
            @(OnClickEvent.HasDelegate ? "" : "disabled")
            @(IsVisible ? "" : "hide")"
            title=@Title @onclick=@OnClickEvent
            @onclick:stopPropagation="true">
            @Text
        </button>
        break;

    case TextButtonType.Default:
        <button class="button button-text
            @(CanPush && IsPushed ? "push" : "")
            @(Disabled ? "disabled" : "")
            @(IsVisible ? "" : "hide")"
            title=@Title
            @onclick=@OnClick
            @onclick:stopPropagation="true">
            @Text
        </button>
        break;

    default:
        break;
}
```

```

@code
{
    [Parameter] public string Text { get; set; } = "";

    [Parameter] public string Id { get; set; } = "";
    [Parameter] public string Source { get; set; } = "#ffffff";

    [Parameter] public string Title { get; set; } = "";

    [Parameter] public TextButtonType ButtonType { get; set; } =
        TextButtonType.Default;

    [Parameter] public bool CanPush { get; set; } = false;
    [Parameter] public bool IsPushed { get; set; } = false;

    [Parameter] public bool IsVisible { get; set; } = true;

    [Parameter] public EventCallback OnClickEvent { get; set; } = null;
    [Parameter] public ITelmaCommand Command { get; set; } = null;

    bool Disabled => Command != null ? !Command.CanExecute(null) : true;

    void OnClick() => Command?.Execute(null);

    protected override void OnParametersSet()
    {
        base.OnParametersSet();

        if (Command != null)
            Command.CanExecuteChanged += (sender, args) =>
                InvokeAsync(() => StateHasChanged());
    }
}

```

ПРИЛОЖЕНИЕ 3. ТЕКСТ ПРОГРАММЫ ДЛЯ РАБОТЫ С ВКЛАДКАМИ TABCONTROL

```
@inherits ComponentBase

<CascadingValue Value="this">
  <Container Classes="header">
    @foreach (TabPage tabPage in Pages)
    {
      if (tabPage == ActivePage)
      {
        <div class="tab-link active-link">@tabPage.Title</div>
      }
      else
      {
        <div class="tab-link" @onclick=@(() => ActivatePage(tabPage))>
          @tabPage.Title
        </div>
      }
    }
  </Container>

  @ChildContent
</CascadingValue>

@code
{
  [Parameter] public RenderFragment ChildContent { get; set; } = null;
  [Parameter] public string ActivePageName { get; set; } = "";

  public TabPage ActivePage { get; set; }
  List<TabPage> Pages = new List<TabPage>();

  public void AddPage(TabPage tabPage)
  {
    Pages.Add(tabPage);
    if (Pages.Count == 1) ActivePage = tabPage;
    StateHasChanged();
  }

  void ActivatePage(TabPage page) => ActivePage = page;

  protected override void OnAfterRender(bool firstRender)
  {
    if (firstRender &&
        Pages.Select(p => p.Title).Contains(ActivePageName))
    {
      ActivatePage(Pages.Find(t => t.Title == ActivePageName));
      StateHasChanged();
    }

    base.OnAfterRender(firstRender);
  }
}
```

ПРИЛОЖЕНИЕ И. ТЕКСТ ПРОГРАММЫ ДЛЯ ОТОБРАЖЕНИЯ ВКЛАДОК TABPAGE

```
@inherits ComponentBase

@if (Parent.ActivePage == this)
{
    <Container Classes="panel panel_show">
        @ChildContent

        <ImageButton ButtonType=@ImageButtonType.Icon
            Class="panel__collapse"
            Title="Collapse panel"
            Image="img/arrow-down.svg" />
    </Container>
}

@code
{
    [CascadingParameter] private TabControl Parent { get; set; }

    [Parameter] public RenderFragment ChildContent { get; set; }
    [Parameter] public string Title { get; set; } = "";

    protected override void OnInitialized()
    {
        if (Parent == null)
            throw new ArgumentNullException(nameof(Parent),
                "TabPage must exist within a TabControl");

        base.OnInitialized();
        Parent.AddPage(this);
    }
}
```

ПРИЛОЖЕНИЕ К. ТЕКСТ ПРОГРАММЫ ДЛЯ ОКОН ПАРАМЕТРОВ PARAMETERS

```
<div class="parameters" id="parameters-@Id">
  <div class="parameters__title">
    <div class="parameters__header">
      @Title
    </div>

    <div class="parameters__buttons">
      <ImageButton ButtonSize=@ButtonSize.Tiny
        Title="Apply"
        Image="img/buttons/selection-form/Ok.svg" />

      <ImageButton ButtonSize=@ButtonSize.Tiny
        Title="Cancel"
        Image="img/buttons/selection-form/Cancel.svg" />
    </div>
  </div>

  <div class="parameters__content">
    @ChildContent
  </div>
</div>

@code
{
  [Parameter] public string Id { get; set; } = "";
  [Parameter] public string Title { get; set; } = "";
  [Parameter] public RenderFragment ChildContent { get; set; } = null;
}
```