Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования «Казанский (Приволжский) федеральный университет»
Институт вычислительной математики и информационных технологий

Кафедра системного анализа и информационных технологий

Направление подготовки: 02.03.02 — Фундаментальная информатика и информационные технологии

Профиль: Системный анализ и информационные технологии

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

РАЗРАБОТКА WEB-ПРИЛОЖЕНИЯ ДЛЯ ПОСТРОЕНИЯ ДИАГРАММ, БЛОК-СХЕМ, ГРАФИКОВ ОНЛАЙН

Обучающийся 4 курса
группы 09-832 (Валиева З.И.)

Руководитель
канд. физ.-мат. наук, доцент (Андрианова А.А.)

Заведующий кафедрой системного анализа
и информационных технологий
д-р техн. наук, профессор (Латыпов Р.Х.)

Казань – 2022

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Современный человек живет в условиях постоянно увеличивающихся информационных потоков. Все более актуальным оказывается умение систематизировать и отсортировывать информацию в рамках рабочего процесса или в ежедневном ритме будничных дел.

Анализ большого объема информации в явном виде является трудозатратой, а иногда даже невыполнимой задачей для человеческого мозга.

Как ускорить процесс восприятия человеком информации? Ответ на данный вопрос очевидно прост. Учеными доказано, что представление информации в графическом виде способствует увеличению скорости его анализа. Согласно статистике, люди запоминают 83% увиденного и только 20% прочитанного.

Человек эффективнее обрабатывает изображение, если ему необходимо обнаружить изменения и сравнить количество, размеры, формы. Быстрота просмотра больших объемов данных зачастую обусловлена сопоставлением свойств символьных данных с визуальными свойствами [1].

Визуализацию информации следует описывать как область, содержащую в себе сочетание различных дисциплин, не имеющих между собой явных связей. Данная область базируется на знании предметной сферы визуализируемых данных и процессов, понимании основ визуального восприятия человеком информации и владения математическими методами анализа данных [2].

Задача упрощения восприятия информации путем графического представления достаточно актуальна на сегодняшний день, несмотря на множество уже разработанных программных средств. С целью быстрого доступа пользователей к инструментам визуализации разрабатываются web-приложения, обратиться к которым можно просто перейдя по ссылке.

Целью работы является разработка web-приложения для построения диаграмм, блок-схем, графиков онлайн. Для достижения цели необходимо реализовать следующие задачи:

1) изучение средств визуализации данных;
2) анализ рынка аналогов создаваемого продукта, выявление их преимуществ и недостатков;
3) проектирование архитектуры приложения;
4) разработка серверной части приложения для реализации бизнес-логики приложения;
5) разработка пользовательского интерфейса;
6) отладка и тестирование продукта.

1. Постановка задачи разработки web-приложения

Визуализация данных – это представление числовой и текстовой информации в виде графиков, диаграмм, структурных схем, таблиц, рисунков, карт и т.д. Данные структуры служат инструментами анализа, на основании результатов которых решаются определенные аналитические задачи, такие как сравнение или выявление причинно-следственной связи, отображение закономерностей или взаимосвязей в данных для одной или нескольких переменных, что делает сложные данные более доступными и понятными.

Преобразование информации в совокупность графических объектов (геометрических примитивов или фигур), другими словами, визуализация данных, используется для передачи данных.

Преимущества визуализации данных представлены на рисунке 1.

Рисунок 1 - Преимущества визуализации данных

Задача разработки web-приложения заключается в создании web-приложения для построения блок-схем, диаграмм, графиков, основным назначением которого является предоставление пользователям возможности создания логического представления процессов в виде блок-схем, визуализации динамики и зависимостей данных в виде графиков, диаграмм с последующей возможностью сохранения результатов. Другими словами, цель разработки состоит в том, чтобы эффективно и в доступном виде передавать информацию пользователям.

1.1. Описание средств визуализации

Для облегчения процесса обработки информации применяются технологии визуализации данных.

Данные могут быть представлены в графическом виде с помощью диаграммы или графика. Описание качественных данных наиболее показательно представляется с помощью диаграммной визуализации. Для отражения зависимости значений одной величины от другой, их сравнения используются графики.

Существует огромное множество видов диаграмм/графиков: линейные, ступенчатые, столбчатые, областные, круговые, пирамидальные диаграммы, гистограммы и др. Выбор типа диаграммы зависит от задачи, которую требуется решить. Например, если стоит задача сравнения показателей, то более эффективным является применение гистограммы, которая использует длину линии для сравнения, а не круговые диаграммы, использующие площадь поверхности для отображения сравнения, так как людям легче обрабатывать различия в длине линии, чем в площади поверхности. Типы диаграмм в виде схемы представлены на рисунке 2.

Рисунок 2 - Типы диаграмм и графиков

Диаграммная визуализации данных широко распространена в маркетинговой, консалтинговой, бухгалтерской областях деятельности.

Для чего нужна визуализация процессов? Представление в графическом виде хода развития события, его детализация во многом упрощают восприятие процесса.

Построение последовательности выполняемых действий - задача, легко реализуемая с помощью блок-схем. Блок-схема представляет ход процесса от события, с которого начинается процесс, до события, которым процесс завершается. Использование блок-схем позволяет разложить процесс на составляющие его действия и более детально рассмотреть их.

Блок-схемы чаще строятся специалистами таких сфер деятельности, как IT, инженерия, строительство.

1.2. Анализ предметной области

Значимость инструментов визуализации данных достаточно высока в связи с тем, что их применение способствует решению задач в работе над чем угодно: от простых линейных диаграмм до сложных комбинированных графиков. В связи с этим основная сложность заключается в выборе инструмента, который обладает всеми желаемыми функциями. Необходимо отметить, что некоторые инструментарии менее многофункциональны, чем другие: одни просты в применении, иные просят продолжительного обучения.

Среди лидеров отрасли на рынке можно выделить Tableau**,** FusionCharts**,** Highcharts. Это платформы, пользующиеся спросом, так как предоставляют обширный диапазон функциональных возможностей для визуализации данных, и при этом имеющие бесплатные версии.

Tableau на сегодняшний день является самым востребованным на рынке продуктом. Его характерными особенностями являются следующие возможности:

— поддержка динамических данных,

— интеграция с другими приложениями с целью обмена большим объемом данных,

— интерактивная графика,

— удобная панель навигации для комплексного управления.

FusionCharts отличается своей легкостью в масштабировании и гибкостью, сопровождается обширной документацией. Ключевой особенностью является информационная панель, включающая в себя мощные инструменты анализа временных рядов, более 85 видов интерактивных графиков и диаграмм.

Highcharts отличается своей простотой, но не уступает аналогам в мощности. Навигация и аналитика ориентированы на клиента. Приложение поддерживает интерактивные карты, которые удобны для использования при работе с географическими данными.

Известными инструментами для работы с блок-схемами является Lucidchart, draw.io, Cacoo. Данные приложения обладают функциями проектирования блок-схем, набором готовых шаблонов, поддерживают организацию совместной работы с другими пользователями, сохранение документов в различных форматах. Необходимо отметить, что Cacoo также обладает функционалом по ведению истории редактирования проекта (при выборе записи из истории выделяется измененный фрагмент, в записи указывается комментарий). В свою очередь посмотреть полную версию проекта с изменения на определенную дату невозможно. Также сервис Cacoo не имеет бесплатной версии.

Популярные на сегодняшний день web-приложения по визуализации данных и процессов оснащены обширными функциональными возможностями. Однако весомым недостатком web-приложений по визуализации данных является отсутствие фиксации изменений для пользователя, т.е. пользователю всегда доступна последняя сохраненная

версия проекта. Добавление возможности вести историю изменения блок-схемы/диаграммы станет полезной опцией для пользователя.

На основании анализа ряда web-приложений для онлайн построения блок-схем/диаграмм можно выделить распространенные базовые функции:

1) на рабочую область из панели объектов имеется возможность перетащить объекты,
2) изменение вида объектов,
3) добавления текста на объекты,
4) настройка вида диаграммы,
5) масштабирование диаграммы,
6) экспорт/импорт данных.

Организация работы пользователя в системе определяется целью, которую он желает достичь. Если пользователь точно знает, что и в каком формате ему необходимо представить графически, то создание проекта в системе необязательно. Однако, если пользователю необходима доработка созданной диаграммы или блок-схемы, то ему следует создать проект, к которому он сможет обращаться в нужный момент.

Для приложения по визуализации данных наиболее значимыми являются функции, упрощающие работу пользователя в системе и при этом способствующие отслеживанию состояния проекта, созданного пользователем.

Выбор из предложенных вариантов шаблона проекта с определенным типом диаграммы/блок-схемы помогает пользователю определиться, какой тип является более подходящим для решения его задачи. Например, если процесс, который собирается описывать пользователь имеет цикличные действия, то безусловно при построении блок-схемы будет необходимо использование конструкций для демонстрации цикла. Выбрав шаблон блок-схемы с циклом при создании проекта, у пользователя на рабочей области уже

будут расположены соединенные блоки, демонстрирующие цикличность процесса.

Помимо шаблонов проектов также имеется возможность использования шаблонов стиля, которые идеально подходят, если пользователю необходимо быстро подобрать стиль диаграммы. Предоставление возможности выбора заранее предопределённого стиля способствует быстрому созданию привлекательной визуализации. В шаблонах стиля настроены цвета, формат текста, расположение заголовка и легенды диаграммы.

Организация многопользовательского совместного доступа реализуется путем предоставления группе пользователей права на просмотр/редактирование проекта, а также путем обработки ситуации одновременного обращения пользователей к проекту.

Ведение истории версий проекта позволяет просматривать и восстанавливать старые версии визуализаций данных, что означает, что пользователь может отменить изменение. Также это удобно, если пользователю необходимо, например, рассмотреть поведение графика при различных данных.

Сочетание ведения истории версий проекта и многопользовательской работы над проектом можно использовать для отслеживания изменений, вносимых разными пользователями. С помощью комментария пользователя при сохранении проекта другие работающие над ним пользователи будут иметь возможность понять, когда и какие изменения были внесены.

2. Проектирование архитектуры web-приложения

Целостность приложения обеспечивается работой каждого компонента web-приложения. В качестве основных структурных компонентов выделяют клиентскую и серверную стороны. Основываясь на логическом распределении web-приложения, обработка логики состоит из организации общения между клиентом и сервером.

Функциональность приложения, с которой взаимодействует пользователь, представляется клиентским компонентом. Серверный компонент, в свою очередь, подразделяется как минимум на две части: логика приложения и хранилище данных. Предназначение первой части заключается в организации управления web-приложением, а второй – хранение всех постоянных данных.

Клиент-серверная архитектура базируется на разделении функций представления, обработки и хранения данных с целью более эффективного использования ресурсов сервера и клиентов.

Компоненты приложения организуют совместную работу путем обмена сообщений в заранее согласованном формате. Структура архитектуры web-приложения представлена на рисунке 3.

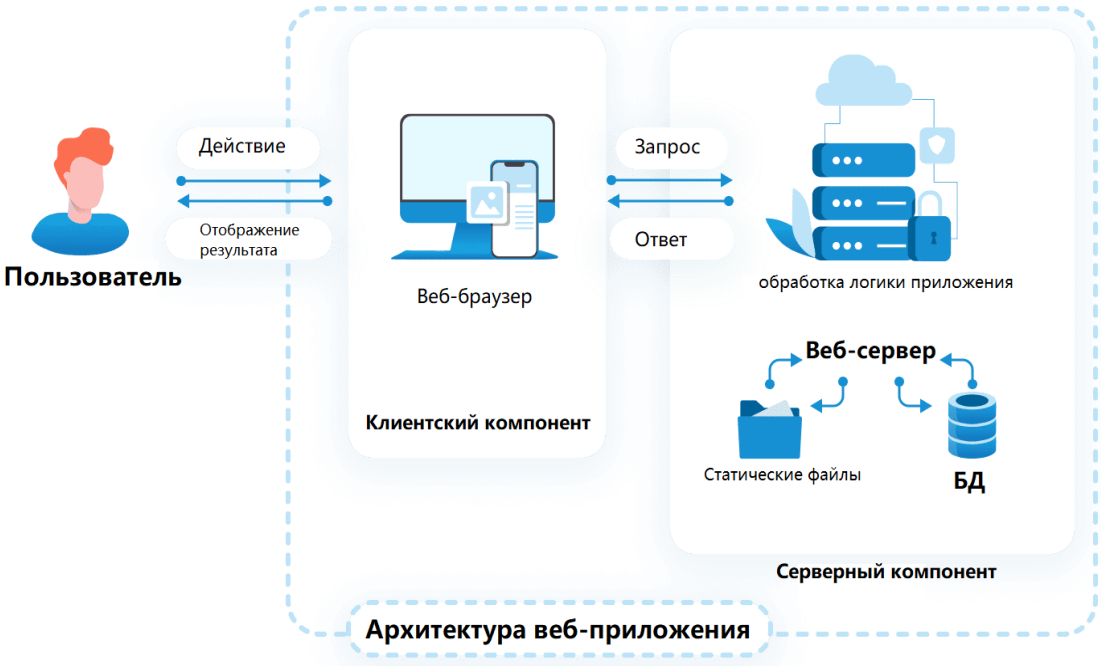

Рисунок 3 - Клиент-серверная архитектура

Роль клиента выполняет web-браузер. Задача сервера заключается в принятии http-запросов от клиентов и выдаче им http-ответов. Под web-сервером подразумевается некая прикладная программа, выполняющая функции web-сервера, однако компьютер, на котором эта программа работает, также называют web-сервером.

База данных фактически не является частью web-приложения. Функционирование базы данных происходит под руководством системы управления базами данных (далее - СУБД). В базе данных хранится вся динамическая информация приложения (учетные, пользовательские данные и др.), свойства которой можно категоризировать.

Шаблон архитектуры сервера базируется на паттерне MVC (Model-View-Controller). Данный шаблон проектирования используется для разделения пользовательского интерфейса (представления), данных (модели) и логики приложения (контроллеры).

Организация работы web-приложения, построенного на модели MVC, представлена на рисунке 4.

Рисунок 4 - Реализация модели MVC в web-приложении

Модель предоставляет доступ к данным. С ее помощью происходит извлечение данные и изменение значений их атрибутов.

Представление служит для отображения данных клиенту в конечном виде (HTML, JSON, …). Пользовательский интерфейс, как правило, создается из данных модели.

Отслеживание различных действий пользователя и их обработка осуществляется контроллером. По заданной логике контроллер оповещает модель о необходимости изменить состояние системы. Другими словами, для обработки запроса клиента происходит обращение контроллера к модели данных, а для формирования и отправки ответа контроллером выбирается представление для рендеринга, отображающее пользовательский интерфейс.

Основная причина, по которой MVC выбирают в качестве шаблона проектирования, заключается в том, что он позволяет разделять аспекты приложения, при этом обеспечивая слабую связь между ними. Разделение каждого вида логики обеспечивает возможность управления сложностью приложения при его создании, так как позволяет акцентировать внимание на одном аспекте реализации за раз.

Слабая связь между тремя основными компонентами приложения MVC также способствует параллельной разработке.

Диаграмма компонентов разрабатываемого web-приложения представлена на рисунке 5.

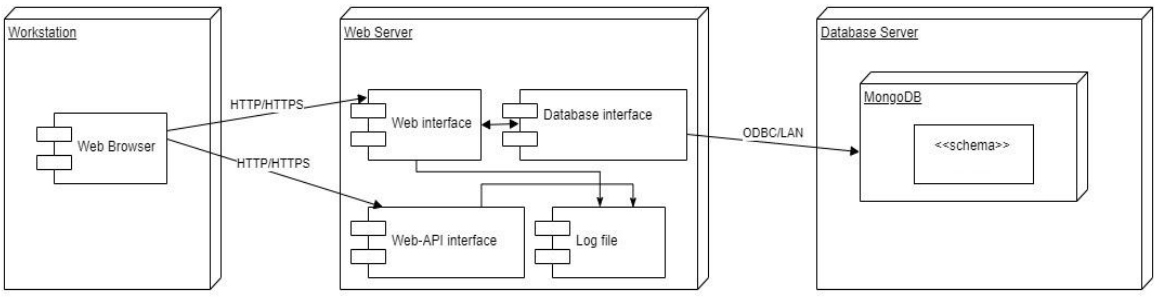

Рисунок 5 - Диаграмма компонентов

Структура проекта приложения подразделяется в соответствии с шаблоном проектирования на 3 блока: «Models», «Controllers», «Views» (рисунок 6).

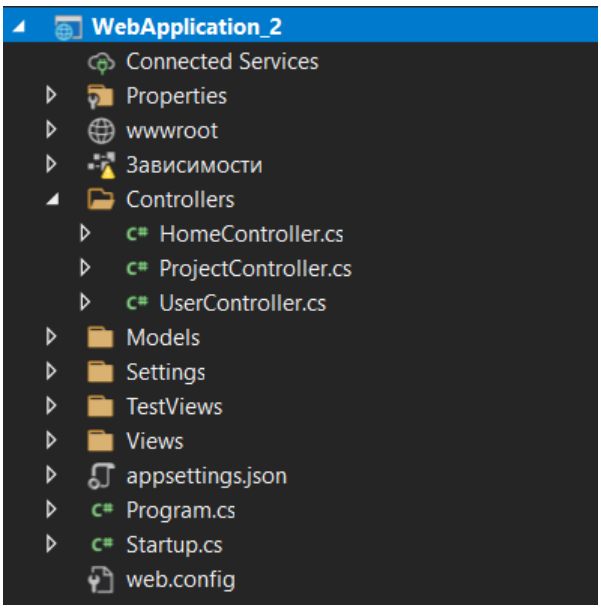

Рисунок 6 – Дерево проекта

Компонент Models содержит описание модели данных пользователя для авторизации/регистрации (User.cs, UserRole.cs), учетных данных пользователя, хранящихся в базе данных (AppUser.cs, AppRole.cs), данных проекта (Project.cs, HistoryProject.cs), точек графика (DataPoint_1.cs, DataPoint_2.cs, DataPoint_3.cs) (рисунок 7). Описание точек графика представлено тремя разными вариантами, так как в зависимости от типа диаграммы, происходит обращение к соответствующей модели.

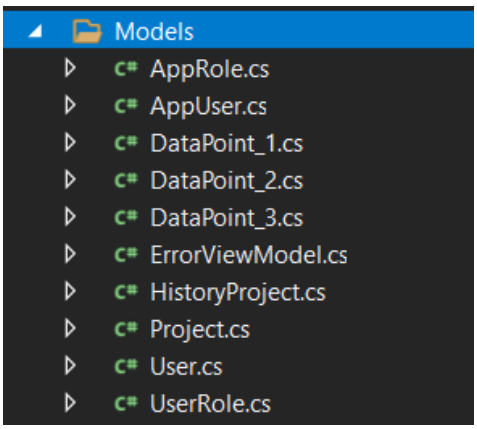

Рисунок 7 – Папка проекта «Models»

Логика приложения при работе пользователя с проектом диаграммы/блок-схемы базируется на обращении к контроллеру ProjectController.cs. В рамках данного контроллера реализованы GET/POST-методы, которые обрабатывают события вызова представления,

переадресации, обращения к коллекциям базы данных, где хранится информация о проекте.

К контроллеру HomeController привязаны загрузка стартовой страницы приложения, обработка ошибок загрузки web-приложения. Методы авторизации/регистрации реализованы в контроллере UserController, где происходит обращение к коллекции базы данных, в которой помещены документы учетных записей пользователей, проектов. Сбор проектов пользователя осуществляется в рамках метода контроллера UserController, который передает в качестве модели в представление Profile.cshtml множество моделей HistoryProject.

Представления разделены по функциональному назначению на те, которые описывают интерфейс проекта, и те, которые отображают интерфейс страниц пользователя (рисунок 8).

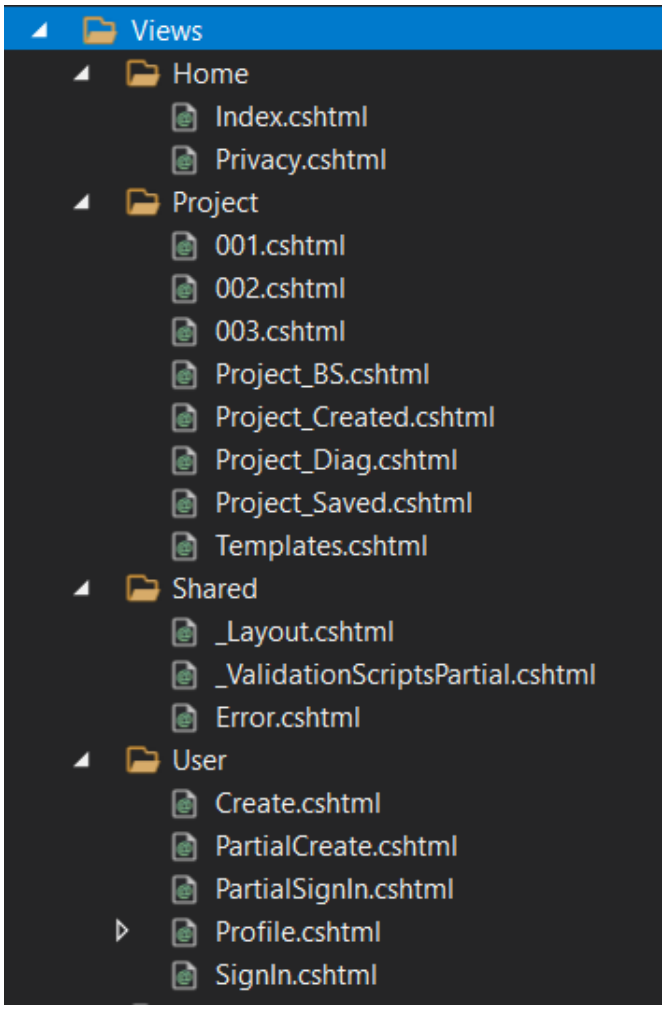

Рисунок 8 – Папка проекта «Views»

Интерфейс для работы с проектом представлен двумя файлами Project_Diag.cshtml (для диаграмм) и Project_BS.cshtml (для блок-схем). Оба файла аналогичны по интерфейсу представления проекта: кнопка «Сохранить», цветовая гамма, выделенный блок рабочей области. Отличие этих двух представлений только в элементах управления, расположенных непосредственно на рабочей области.

Представление Templates.cshtml реализует интерфейс демонстрации шаблонов блок-схем, диаграмм.

Файлы Project_Created.cshtml и Project_Saved.cshtml используются как частичные представления, возвращаемые при ajax-запросах на создание и сохранение проекта блок-схемы/диаграммы. В них определены модальные окна, которые содержат сообщение об успешности создания/сохранения проекта.

Представления 001.cshtml, 002.cshtml, 003.cshtml также используются в качестве частичных представлений. Они отображают диаграмму на web-странице. Представление 001.cshtml отрисовывает линейные, столбчатые, штабелированные диаграммы, гистограммы. Представление 002.cshtml позволяет отобразить областные, круговые, пирамидальные диаграммы. Представление 003.cshtml способствует построению финансовых диаграмм.

Представления пользовательских страниц включают в себя окно регистрации (SignIn.cshtml), авторизации (Create.cshtml). Файл Profile.cshtml реализует интерфейс отображения истории изменения проектов, доступных пользователю.

3. Стек технологий

Разработка в ASP.NET предоставляет все необходимые сервисы для создания устойчивых web-приложений, сочетание которых образует различные модели разработки [3].

На сегодняшний день популярно использование платформы ASP.NET Core для разработки web-приложения. Данный фреймворк является модификацией ASP.NET 4.x. Изменения в архитектуре ASP.NET Core соответствуют современным требованиям и методам разработки, делая структуру создаваемого программного продукта более модульной и надежной. Кроссплатформенность ASP.NET Core также является весомым преимуществом. Разработку с открытым исходным кодом (Windows, Linux и Mac) можно использовать для развертывания в облачной и локальной среде.

Платформой для разработки web-приложения является ASP.NET Core MVC [4]. Фреймворк ASP.NET Core MVC перенял все преимущества ASP.NET Core:

1) полная независимость web-сервера,
2) кроссплатформенная разработка,
3) открытый исходный код,
4) управление версиями приложений,
5) унифицированный стек для MVC и Web API.

3.1. Серверная часть

Для разработки web-приложения используется язык программирования C#. C# реализуется на платформе .NET (многоязыковая среда). Данная платформа в рамках разработки базируется на каркасе .NET Core. Каркас .NET Core имеет большую библиотеку классов. Библиотека .NET включает множество интерфейсов и классов, объединенных в группы по тематике. Пространство имен (namespace) задает каждую группу, корневым пространством имен является System. Между классами библиотек прослеживаются отношения наследования.

Среди преимуществ web-приложений на C# необходимо выделить такие свойства, как кроссплатформенность, мультифункциональность, надёжность [5].

3.2. Клиентская часть

Пользовательский интерфейс приложения реализуется клиентской частью. Интерфейс представляется в виде динамических web-страниц. Возможность быстро создавать интерфейс для организации взаимодействия с данными (запросы, обновления) содержится в ASP.NET Core MVC. Создание формы происходит динамически на основе типизированной модели данных. Правила проверки в html-форме определяются с помощью атрибутов C#, применяемых как на клиенте, так и на сервере.

Для написания интерфейса основным компонентом стека технологий является язык разметки HTML. Использование html-элементов позволяет разделять страницу на области определенного назначения (шапка страницы, меню и др.), описывать изображения, таблицы, формы ввода данных и т.д. При отправке браузером запроса url-адреса для получения страницы сначала в качестве отчета приходит html-документ. Данный документ содержит дополнительные сведения об оформлении или макете в виде CSS, а также о поведении в виде JavaScript [6].

Язык таблицы стилей CSS используется для обработки внешнего вида страницы. Обращаясь к свойствам объектов, разработчик может управлять цветом текста, стилем шрифтов, расстоянием между абзацами, размером и расположением столбцов, использованием фоновых изображений или цветов, дизайном макетов, вариациями отображения для разных устройств и размеров экрана, а также множеством других эффектов.

Создание интерактивности элементов web-страниц осуществляется путём написания функций на языке JavaScript. События, инициированные пользователем (нажатие на кнопки, загрузка изображения и др.), возможно

перехватить с помощью JavaScript, тем самым это поспособствует снижению нагрузки на сервер, так как трафик сервера будет сэкономлен.

3.3. База данных

В качестве СУБД используется нереляционное хранилище данных MongoDB [7].

MongoDB представляет собой документо-ориентированную систему управления данными, которая является при этом кроссплатформенной. MongoDB отличается от традиционной реляционной структуры базы данных тем, что данные хранятся в виде json-подобных документов с динамическими схемами. Это отличие делает интеграцию данных в определенных видах приложений проще и быстрее.

Данные хранятся как документы в двоичном представлении. Такой документ называется BSON [8]. В BSON-документах создаются поля, и каждое поле содержит значение определенного типа данных, к ним относятся также массивы, бинарные данные.

Проведение аналогии с реляционными базами данных позволяет отметить, что вместо таблиц и строк в MongoDB определены коллекции и документы. Документ состоит из пар «ключ — значение». Коллекцией, в свою очередь, называют набор документов. Документы в рамках одной коллекции могут отличаться друг от друга размером, содержанием и количеством полей.

На рисунке 9 представлен интерфейс коллекции и документов в базе данных MongoDB.

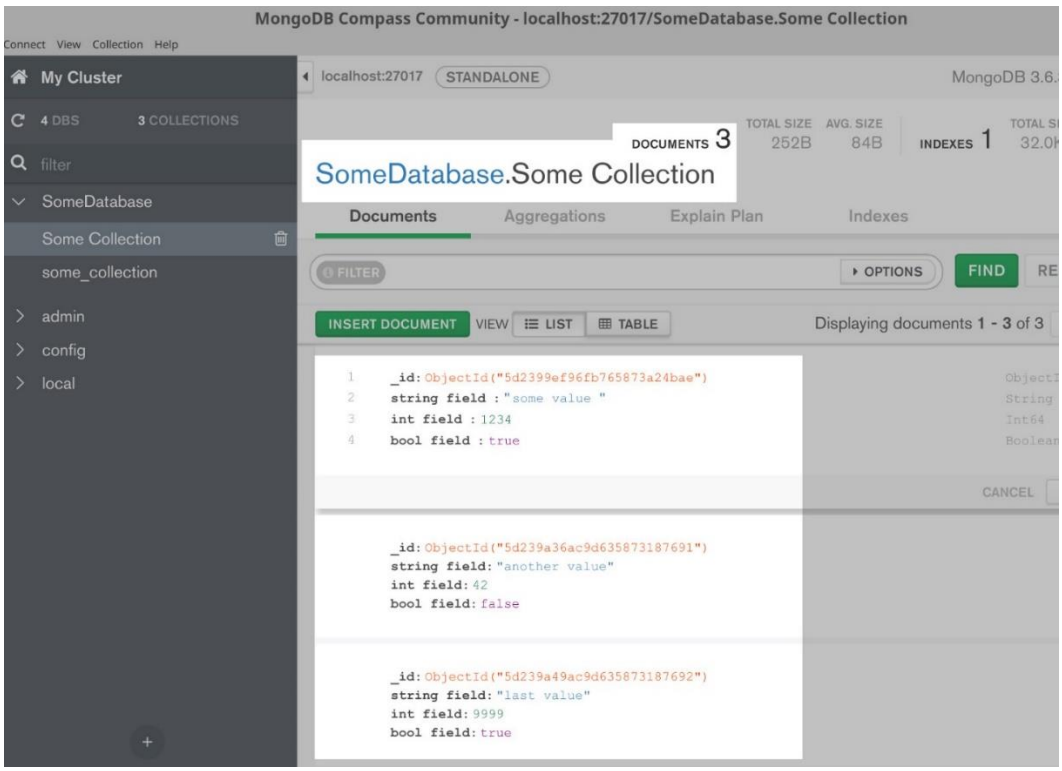

Рисунок 9 - Вид документа

Для того, чтобы в ASP.NET Core MVC была возможна работа с базой данных MongoDB, необходимо предварительно добавить к проекту драйвера (пакет MongoDB.Driver).

Библиотека MongoDB.BSON.dll организует работу по сопоставлению документов BSON с объектами классов на C#.

В библиотеке MongoDB.Driver.Core.dll определен функционал для установки соединения с сервером.

Библиотека MongoDB.Driver.dll представляет, так называемую, обертку для взаимодействия кода на C# с сервером MongoDB.

4. Разработка приложения

4.1. Реализация работы с блок-схемами

Базовый функционал web-приложения подразделяется на две группы: функции для построения блок-схем и функции для работы с графиками/диаграммами.

Создание блок-схемы подразумевает логическое расположение на рабочей области блоков и их соединение между собой в связные компоненты, причем с возможностью добавления текста/комментариев. Вид проекта для построения блок-схем представлена на рисунке 10.

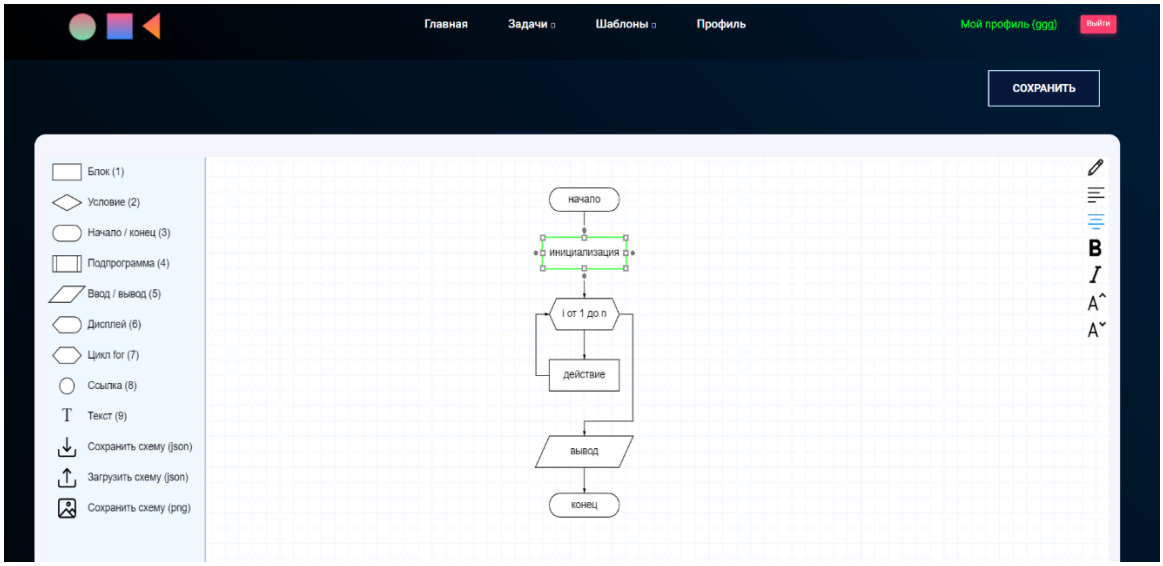

Рисунок 10 – Проект блок-схемы

Выгрузка блок-схемы осуществляется в двух форматах: png и json. Загрузка блок-схемы происходит в формате json с полным очищением текущего содержимого рабочей области.

Рабочая область для построения блок-схемы описывается элементом <canvas> [9]. В рамках данного элемента определена стандартная система координат Cartesian, с начальной точкой (0, 0), расположенной в верхнем левом углу элемента. При движении вправо значение координаты расположения объекта по оси X будет увеличиваться, при движении вниз – по оси Y (рисунок 11).

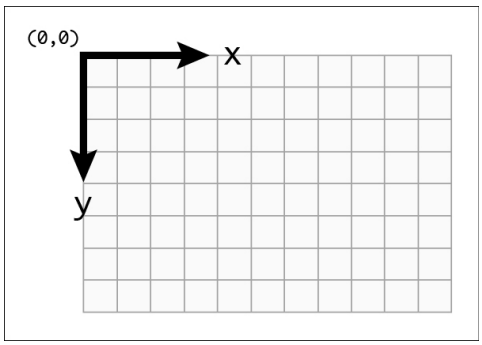

Рисунок 11 – Координатная плоскость

Для работы с холстом следует сначала получить объект холста с помощью метода document.getElementById() языка JavaScript. Следующим шагом является получение двумерного контекста рисования. Решить задачу возможно с помощью метода getContext(). Контекст рисования необходимо получить для того, чтобы в дальнейшем была возможность использовать методы рисования простых двумерных фигур (контуры, прямоугольники, дуги). Сразу же после загрузки страницы вызывается метод для получения объекта контекста.

Разработка функционала по построению блок-схемы базируется на прототипном наследовании JavaScript [10]. Механизм, организующий наследование свойств друг друга объектов JavaScript, называется прототипирование. Для каждого объекта определен свой прототип, который, в свою очередь, представляет собой некий шаблон, от которого объект наследует свойства и методы. Также имеется возможность построения цепочки прототипов, которая строится путем определения прототипу еще и своего прототипа. Таким образом можно получить ситуации, когда одним объектам доступны свойства и методы, которые определены в других объектах.

Prototype - это встроенный объект, функции которого доступны, изменяемы и могут создавать новые переменные и методы для прототипа (рисунок 12).

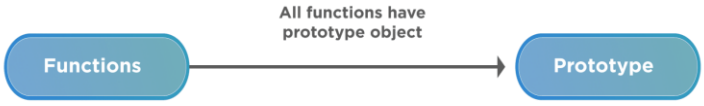

Рисунок 12 – Взаимосвязь функции и прототипа

В js-файле определены следующие функции и их объекты-прототипы:

1) Connector – функция, определяющая точку соединения, которая описывается с помощью блока и координаты соединения. В качестве прототипов функции объявлены рисование (Draw), расчет дистанции (GetDistance), перемещение курсора мыши по рабочей области (IsMouseHover), обновление в случае изменения размера блока (Update).

2) Block – функция, задающая описание блока посредством координаты левого верхнего угла, ширины, высоты, надписи, типа блока. Прототипами функции являются события, фиксирующие текст надписи на блоке по умолчанию, обновление размеров блока при увеличении размера текста (в ширину/высоту), рисование различных типов блоков (действие, условие, цикл, ввод/вывод и др.), инициализацию соединителей (верхний, нижний, правый, левый), выделение элементов, перемещение, копирование.

3) Arrow – функция, описывающая непосредственно линию соединения двух блоков. Прототипы функции определяют построение стрелки, расчет оптимальной траектории прокладывания линии соединения.

4) TouchEvents – функция, которая принимает в качестве аргумента схему. Прототипы функции задают описание событий перемещения блок-схемы по рабочей области: расчет расстояния сдвига, выделение схемы, сдвиг.

5) Diagramm – функция, которая определяет взаимодействие с рабочей областью в целом. В качестве прототипов функции выступают инициализация объектов панели меню и обработка их действий, события добавления блоков, стрелок, текста на макет, изменение размера объектов, выделение/копирование/удаление элементов.

Блок-схема хранится в базе данных внутри документа о проекте в поле data в виде массива. В массиве отдельно описываются блоки (blocks),

соединительные стрелки (arrows) и координаты левого верхнего угла блок-схемы (рисунок 13).

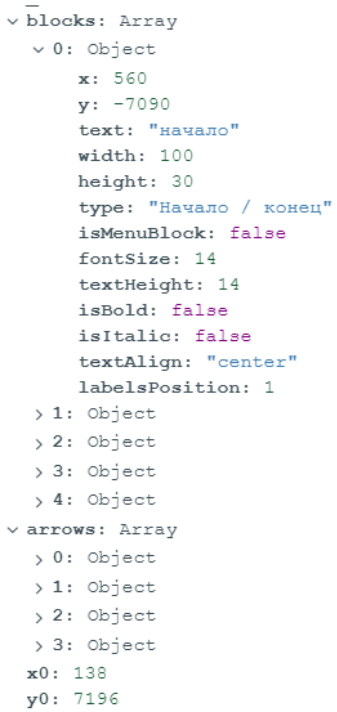

Рисунок 13 – Описание блок-схемы в базе данных

В качестве шаблонов блок-схем представлены линейная схема (последовательное выполнения операций (команд, указаний), то есть выполнение действий происходит друг за другом), циклическая (многократное повторение определенной последовательности действий) и разветвляющаяся (выполнение хотя бы одной операции по проверке условия, в результате чего осуществляется переход действия на какой-нибудь другой из возможных вариантов решения). Страница шаблонов блок-схем представлена на рисунке 14.

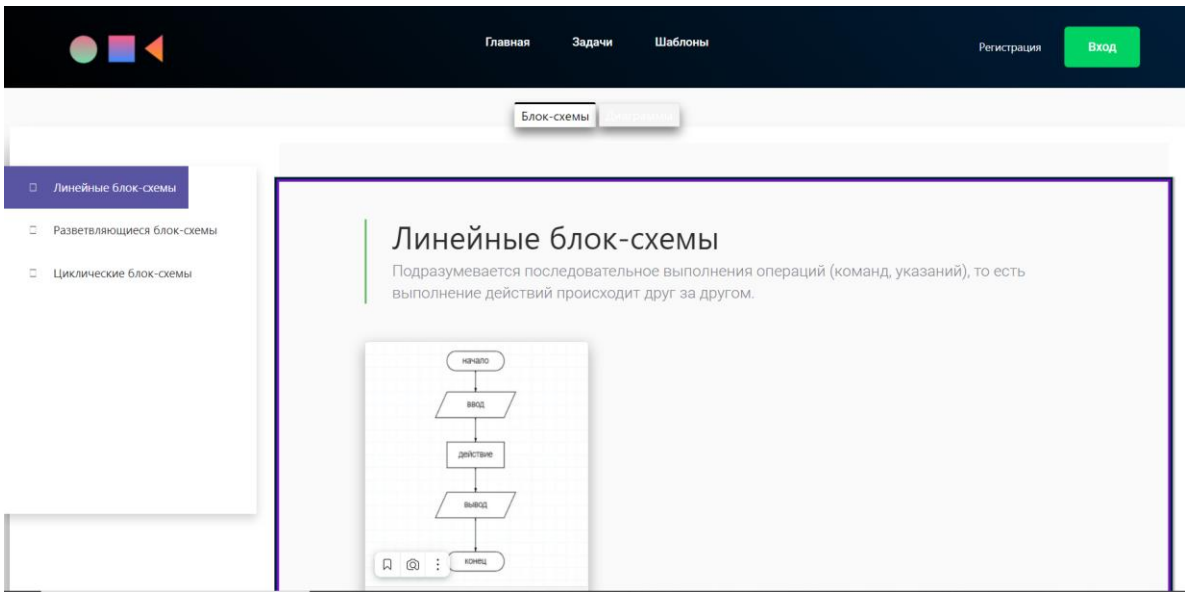

Рисунок 14 – Страница шаблонов блок-схем

Работа с элементами блок-схемы организована следующим образом:

— после двойного щелчка по блоку открывается возможность добавления или редактирования текста блока (рисунок 15);

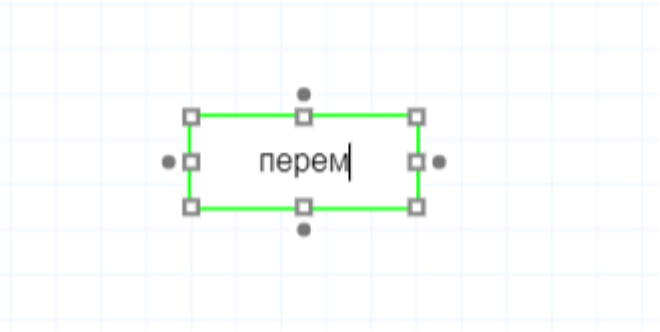

Рисунок 15 – Изменения текста надписи

— при увеличении объема текста размер блока масштабируется горизонтально или вертикально (рисунок 16);

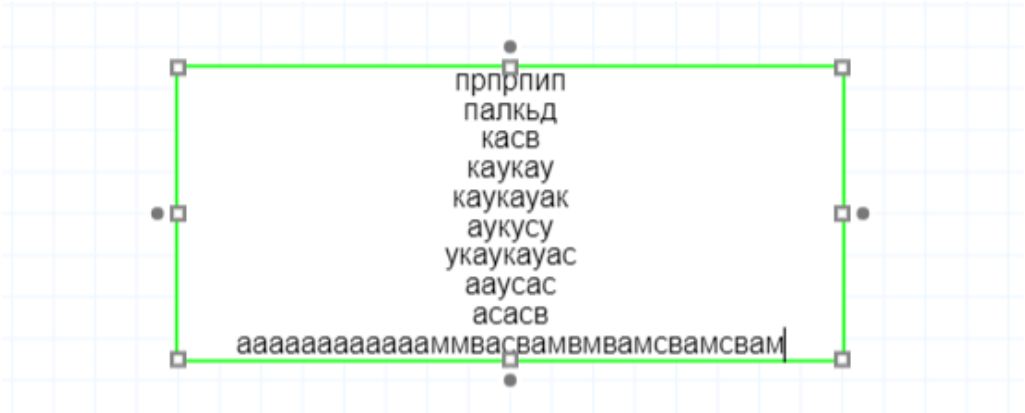

Рисунок 16 – Изменение размера блока

— во время редактирования текста возможно выбирать стиль (курсив, полужирный), увеличивать размер шрифта (рисунок 17);

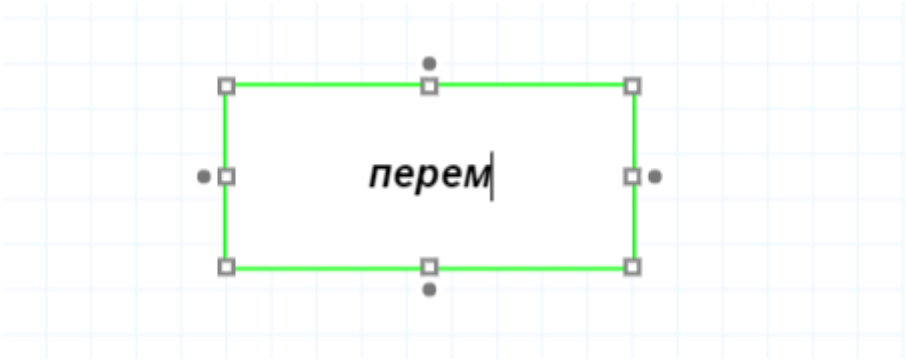

Рисунок 17 – Редактирование стиля надписи

— линия соединения блоков при перемещении одного из блоков автоматически перестраивает свою траекторию (рисунок 18);

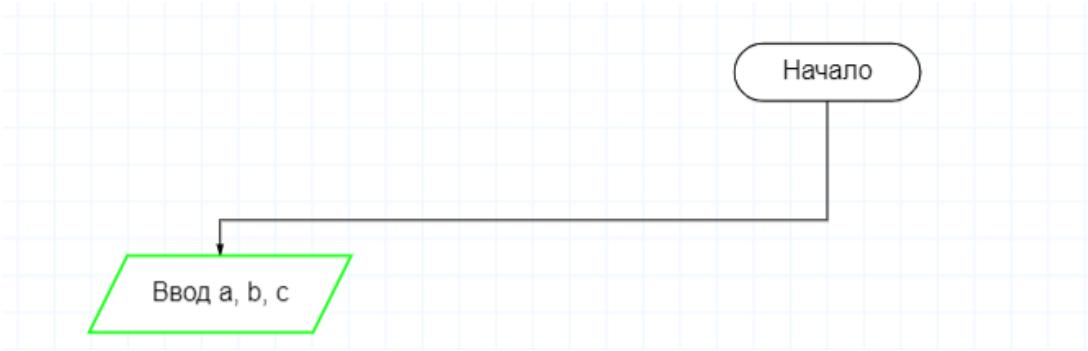

Рисунок 18 – Линия соединения

— на рабочую область возможно добавление объектов путем копирования уже имеющихся на холсте объектов.

Реализованный функционал поддерживает построение вложенных процессов с добавлением комментариев к фрагментам блок-схем (рисунок 19).

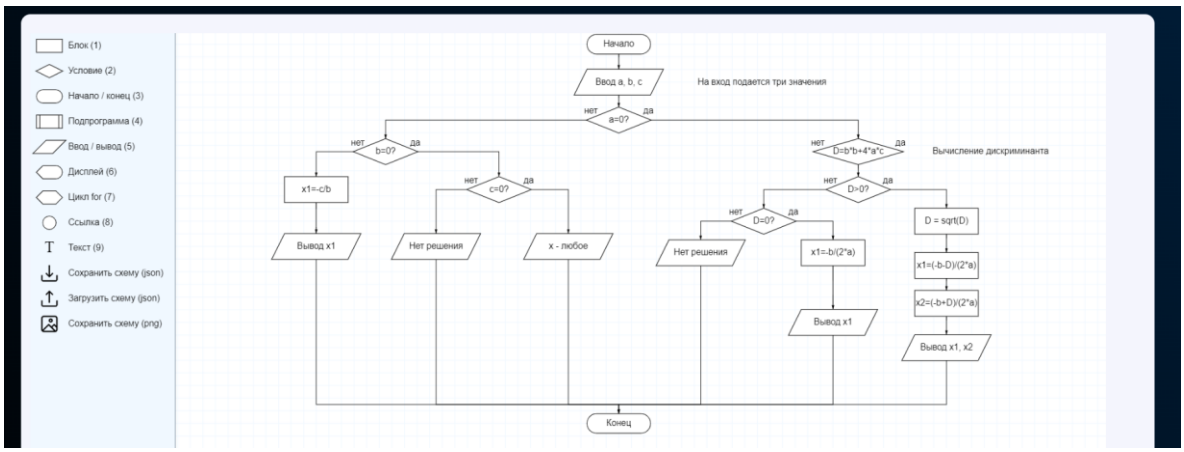

Рисунок 19 – Разветвляющаяся блок-схема

4.2. Реализация работы с диаграммами

Построение диаграммам/графиков подразумевает работу с данными, которые следует визуализировать, и с представлением, т.е. с тем, как и в каком виде представить данные.

Для создания графического образа диаграмм используется библиотека JavaScript – CanvasJS [11]. Возможности библиотеки описываются такими функциями, как всплывающая подсказка, масштабирование, детализация, поддержка более 25 различных типов диаграмм и др. Сочетание возможностей JavaScript и библиотеки CanvasJS позволяет создавать интерактивные диаграммы:

1) при наведении на точки данных появляется информация о ней,

2) соответствующие элементы из легенды диаграммы при выборе подсвечиваются.

Также преимущество CanvasJS заключается в том, что библиотека поддерживает интеграцию с Bootstrap и несколькими другими фреймворками и библиотеками JavaScript.

Работа с данными реализуется посредством методов jQuery [12]. В web-приложениях, которые зависят от динамической связи клиент-сервер, целесообразно использование ajax-запросов. Асинхронный http-запрос обновляет части web-страниц без перезагрузки или обновления всей страницы. Это достигается за счет использования возможностей JavaScript для организации асинхронного запроса и получения в качестве ответа json-данных.

В качестве основных преимуществ использования ajax выделяют следующие:

1) из-за уменьшения объёма передаваемых данных между клиентом и сервером снижается количество затрачиваемого трафика;

2) генерация только части страницы способствует уменьшению нагрузки на сервер;

3) из-за того, что нет необходимости в перегрузке всей страницы, скорость отклика на действие пользователя заметно уменьшается;

4) за счет увеличения быстродействия повышается интерактивность страницы, что позволяет сделать ресурс более удобным для пользования.

Применение асинхронных ajax-запросов в рамках работы с диаграммами обусловлено, прежде всего, необходимостью поддерживать данные в ходе всего процесса работы с ними. Если при ответах сервера на запросы клиента страница будет обновляться, то используемые пользователем данные потеряются.

Вид проекта для построения диаграмм/графиков представлен на рисунке 20.

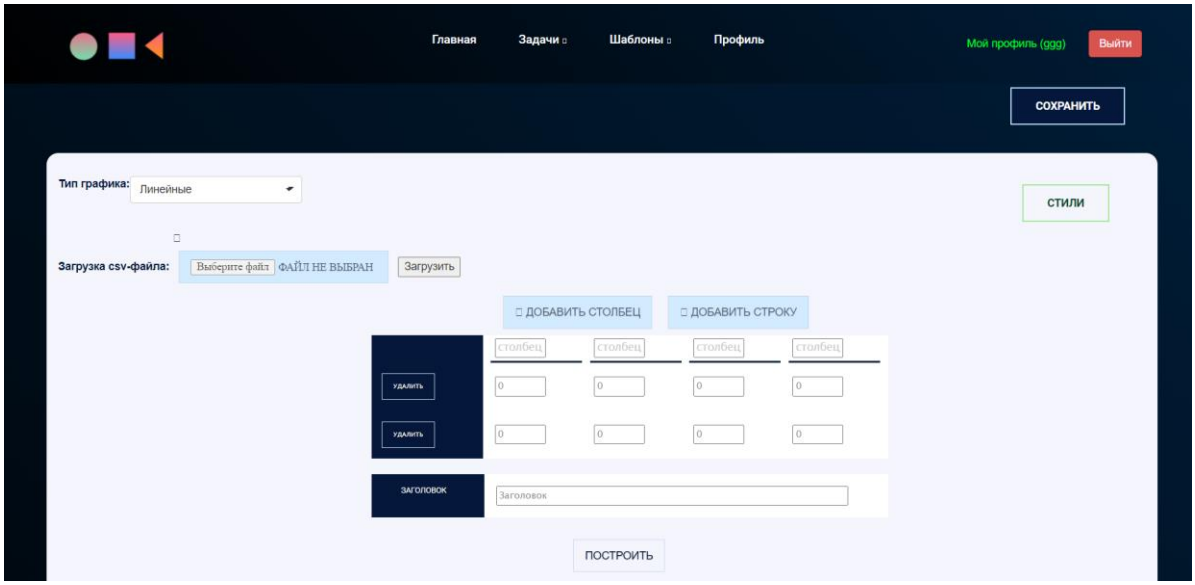

Рисунок 20 – Проект создания диаграммы

В приложении предусмотрен выбор шаблона диаграммы при создании проекта. В качестве шаблонов диаграмм представлены различные типы: линейные, финансовые, областные, круговые, пирамидальные, комбинированные и другие. Страница с шаблонами продемонстрирована на рисунке 21.

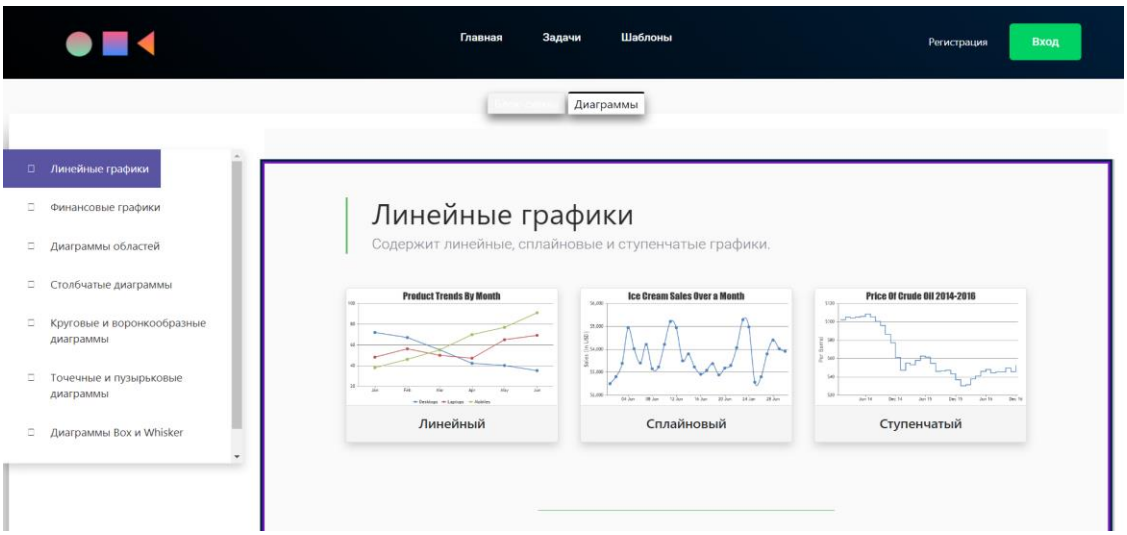

Рисунок 21 – Шаблоны диаграмм

При создании диаграммы пользователь имеет возможность настройки данных: добавление столбцов/строк, ввод заголовка диаграммы, загрузка данных с компьютера. Для импорта данных для диаграммы/графика поддерживается формат csv (рисунок 22). При загрузке csv-файла данные на страницы отображаются в виде таблицы. Реализация загрузки csv-файла заключается в определении в html элемента интерфейса <input> с типом file [13]. Обработка отображения данных файла в таблице организована с помощью функции JavaScript, срабатывающей при нажатии на кнопку «Загрузить». Если файл невозможно считать, то будет выведено оконное сообщение «Please upload a valid CSV file.».

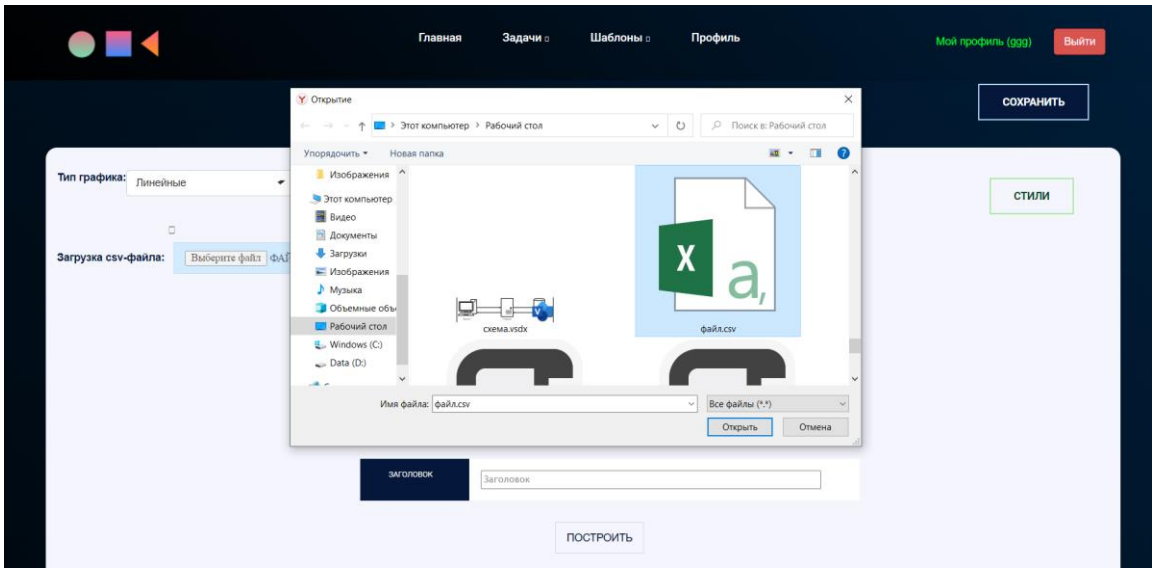

Рисунок 22 – Загрузка csv-файла

Чтобы выгрузить диаграмму на компьютер, пользователю необходимо нажать на кнопку «Сохранить как картинку», которая находится под диаграммой, и начнется скачивание диаграммы или графика в формате png (рисунок 23).

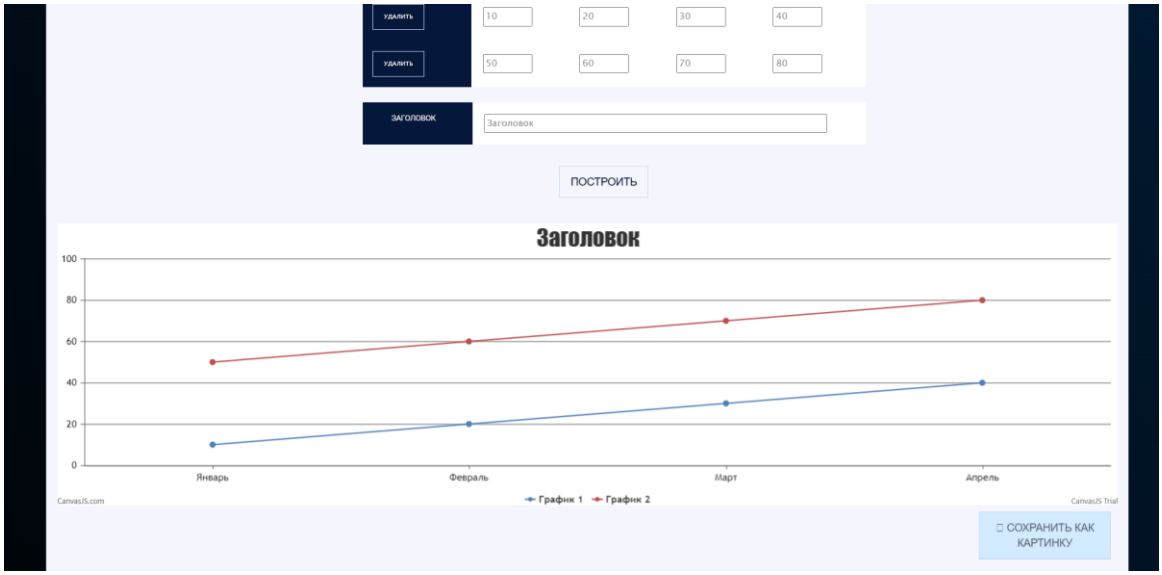

Рисунок 23 – Сохранение диаграммы в виде картинки

Пользователю доступно изменение типа диаграммы, если изначально был выбран другой тип. Выбор типа диаграммы осуществляется через элемент интерфейса в виде раскрывающегося списка (рисунок 24). После выбора нового вида диаграммы необходимо заново нажать кнопку «Построить».

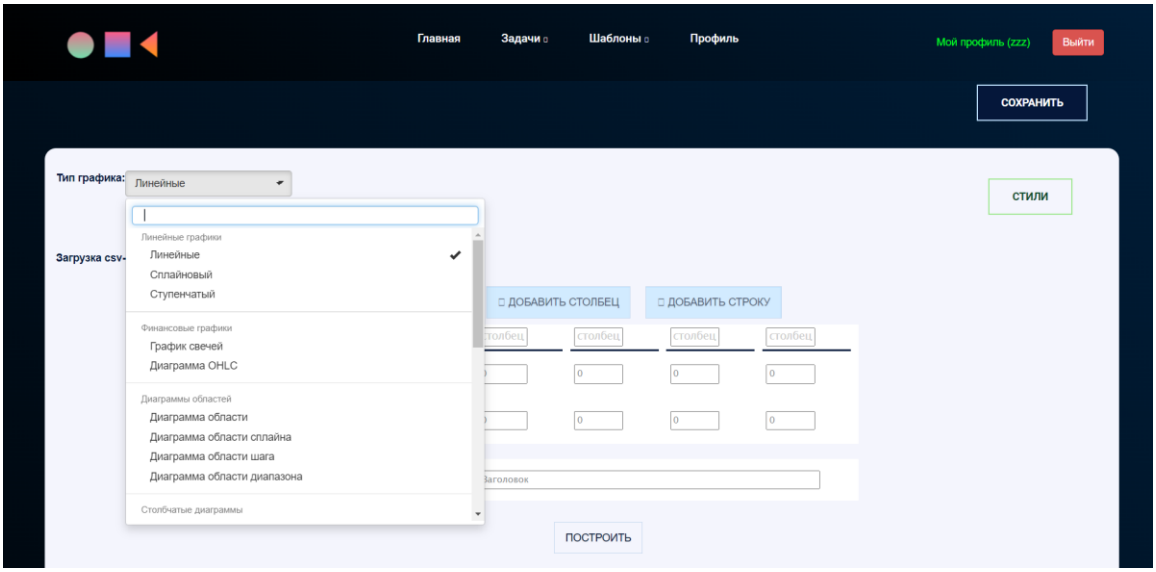

Рисунок 24 – Смена типа диаграммы

Пример круговой диаграммы представлен на рисунке 25.

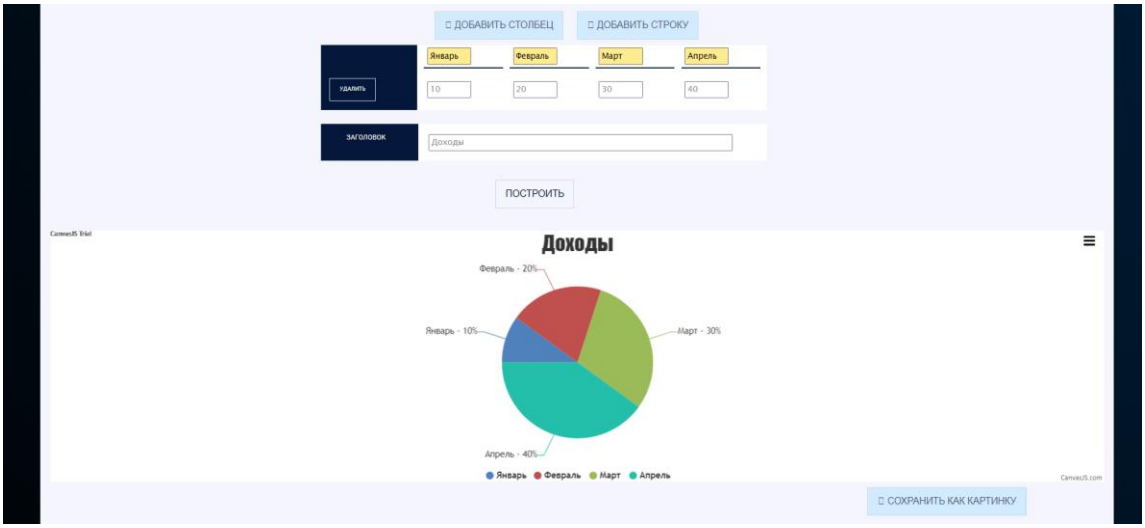

Рисунок 25 – Круговая диаграмма

Ввод данных для таких диаграмм, как, например, диаграмма диапазонов, пузырьковая, осуществляется в рамках одной строки через запятую (рисунок 26, рисунок 27).

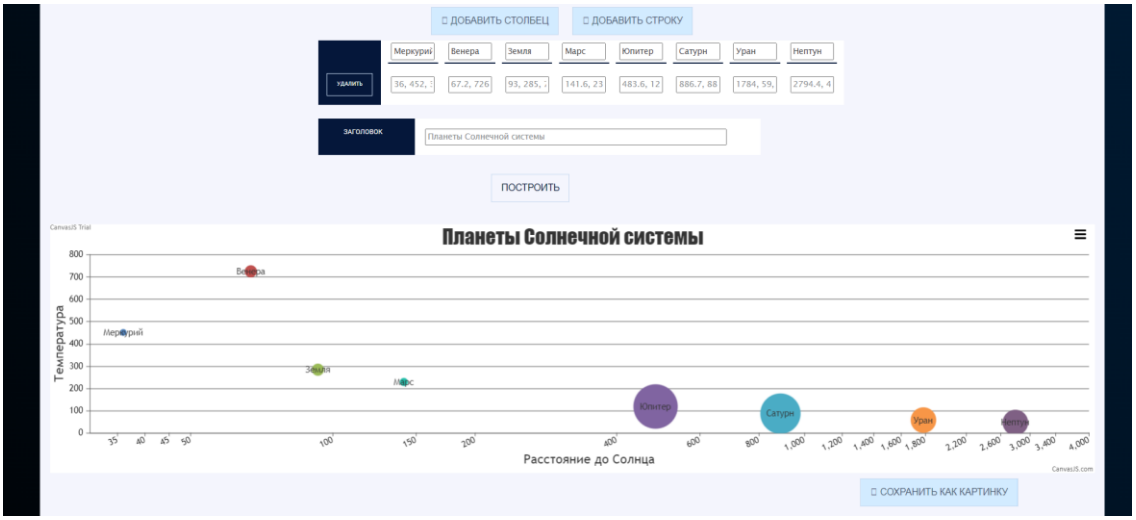

Рисунок 26 – Пузырьковая диаграмма

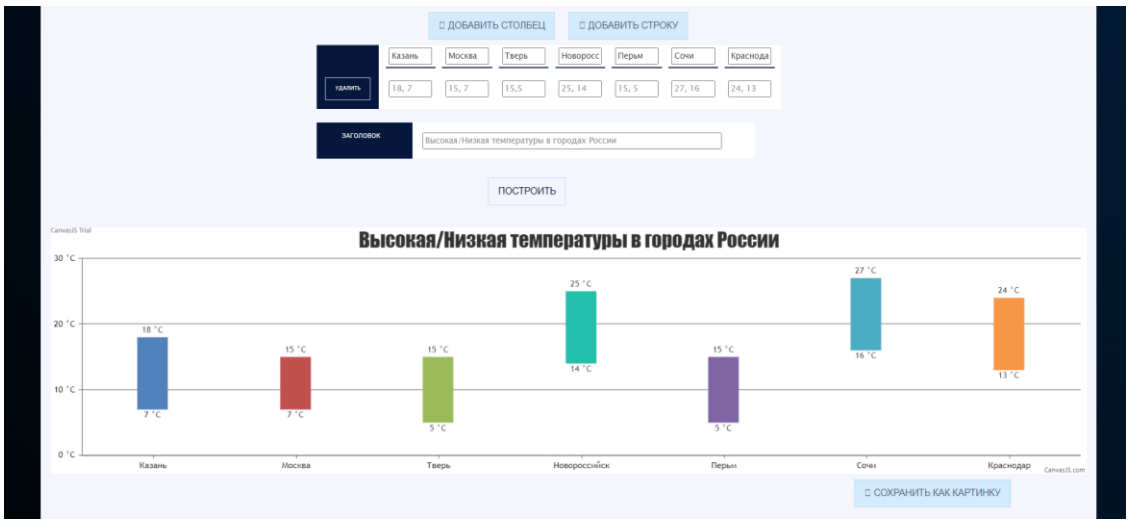

Рисунок 27 – Диаграмма диапазона

Работу над оформлением диаграммы облегчает наличие шаблонов стиля. Стилевое оформление диаграммы определяется выбором трех базовых цветов и расположения заголовка и легенды. Базовые цвета используются по порядку, например, красный для первой линии графика или области, синий – для второй и т.д. Далее происходит смешивание цветов. Заголовок и легенда могут располагаться по одну сторону от диаграммы или в разных углах, это зависит от выбранного шаблона (рисунок 28).

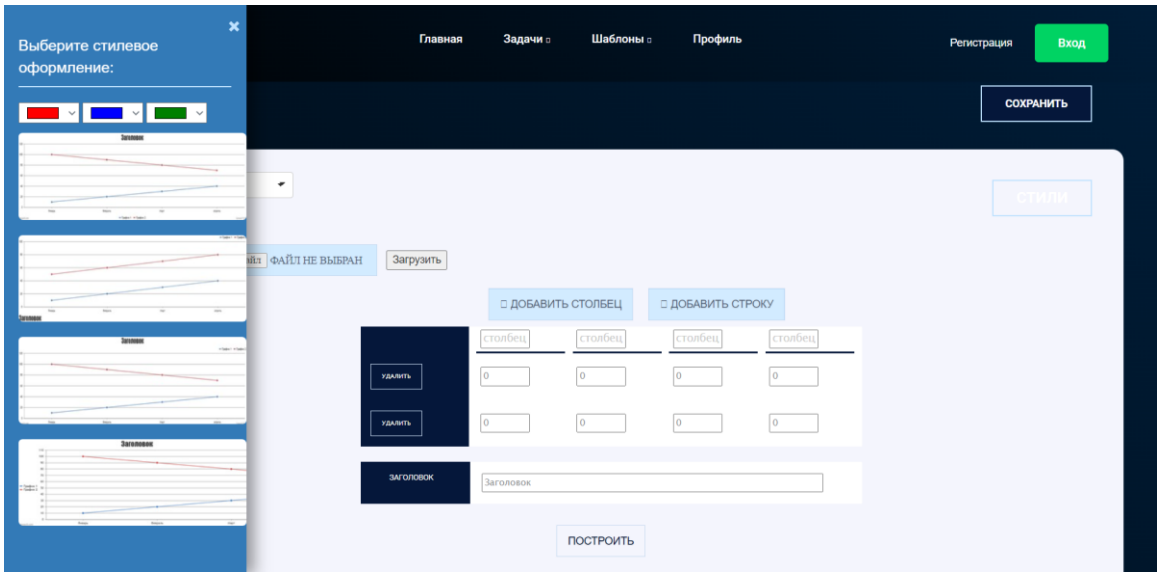

Рисунок 28 – Настройка стиля диаграммы

4.3. Реализация истории изменения проекта

Пользователи в системе подразделяются на авторизованных и неавторизованных. В соответствие с ролью функционал, доступный для пользователя, различается.

ASP.NET Core Identity представляет встроенную в ASP.NET Core систему аутентификации и авторизации. Данная система позволяет пользователям создавать учетные записи, аутентифицироваться, управлять учетными записями.

Для работы с данными пользователя создана модель «User», в которой определены поля логин пользователя, адрес электронной почты, пароль.

Функционал аутентификации и авторизации на основе системы Identity становится доступным через вызовы методов UseAuthentication и app.UseAuthorization() в файле Startup.cs.

Данные о пароле пользователя в базе данных хранятся в хешированном виде. ASP.NET Core позволяет хешировать пароль с использованием алгоритма PBKDF2.

Страница авторизации представлена на рисунке 29.

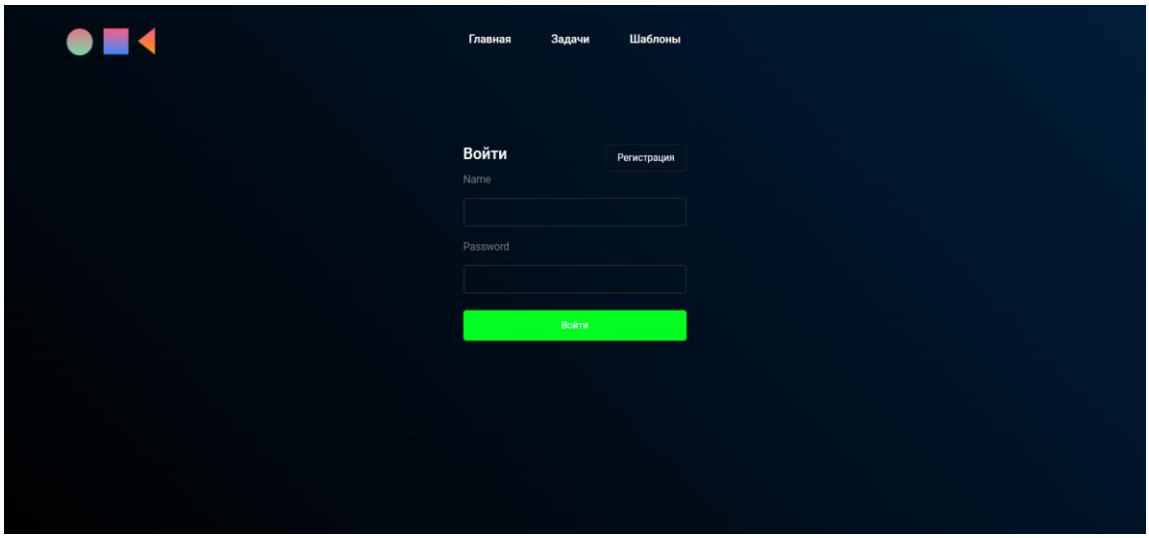

Рисунок 29 - Страница авторизации

Основным функциональным отличием авторизованного пользователя от неавторизованного является возможность сохранения проекта с целью последующего возвращения к работе над ним. В профиле пользователя

отображается история работы над проектом, где фиксируются изменения, а также есть возможность обратиться к виду проекта на момент добавления изменения.

При работе с проектом в случае обновления страницы все данные очистятся. Для исключения подобной ситуации ответ сервера на действия клиента в рамках проекта реализуется через ajax-запросы.

Для хранения проектов в базе данных MongoDB реализованы две коллекции: Projects и HistoryProjects. Коллекция Projects предназначена для хранения документа о созданном проекте. Документ содержит поля имя проекта, автора проекта и список пользователей, которым доступен проект, и логин активного пользователя (рисунок 30).

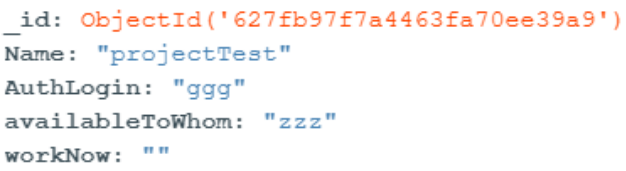

Рисунок 30 – Документ в коллекции Projects

Коллекция HistoryProjects служит для хранения документов об изменении проекта. В документе определены наименование проекта, логин автора проекта, автора изменений, данные, стиль, дата сохранения, комментарий пользователя (рисунок 31). Она также содержит документ о создании проекта, где в качестве комментария автоматически указывается «Проект создан. Наименование проекта: …».

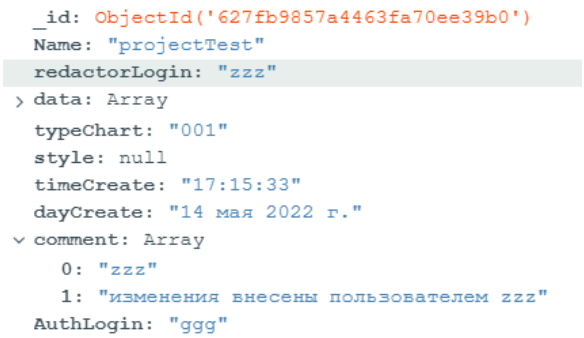

Рисунок 31 – Документ в коллекции HistoryProjects

Для работы над проектом визуально выделена область на странице, где размещены элементы интерфейса для построения диаграммы/графика, либо для проектирования блок-схемы, и помещена кнопка «Сохранить».

Кнопка «Сохранить» вызывает модальное окно с параметрами для сохранения проекта. Возможность сохранения проекта доступна только авторизованным пользователям. При попытке неавторизованного пользователя сохранить проект в модальном окне выводится сообщение с просьбой войти в систему (рисунок 32).

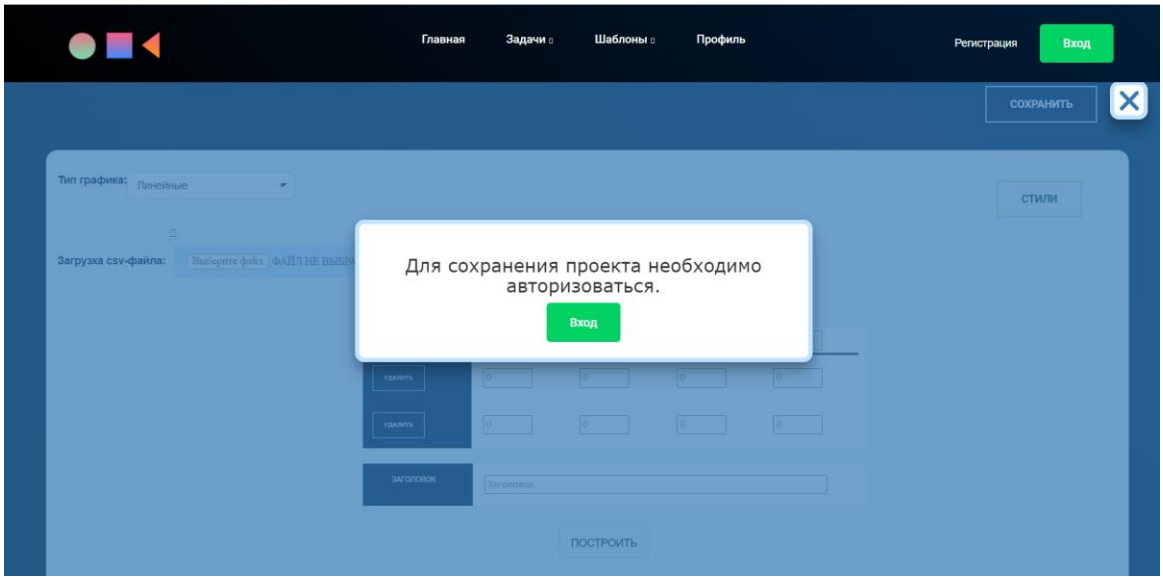

Рисунок 32 – Сообщение о необходимости авторизации

В рамках проекта без перенаправления на страницу авторизации пользователь может войти в систему или в случае отсутствия учетной записи зарегистрироваться (рисунок 33).

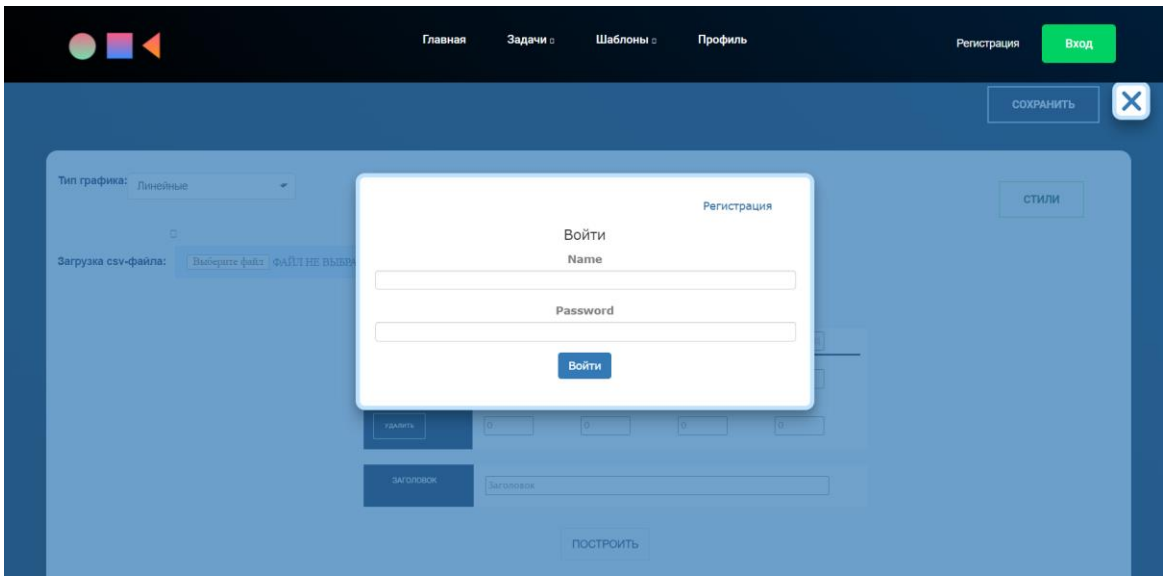

Рисунок 33 – Модальное окно авторизации

При создании проекта пользователю необходимо ввести наименование проекта и указать, кому будет доступен проект (рисунок 34). Создатель проекта имеет возможность открыть доступ к проекту другим пользователям. Автору проекта необходимо добавить логин пользователя в список тех, кому доступна работа над проектом. В результате создания проекта выводится сообщение «Проект создан».

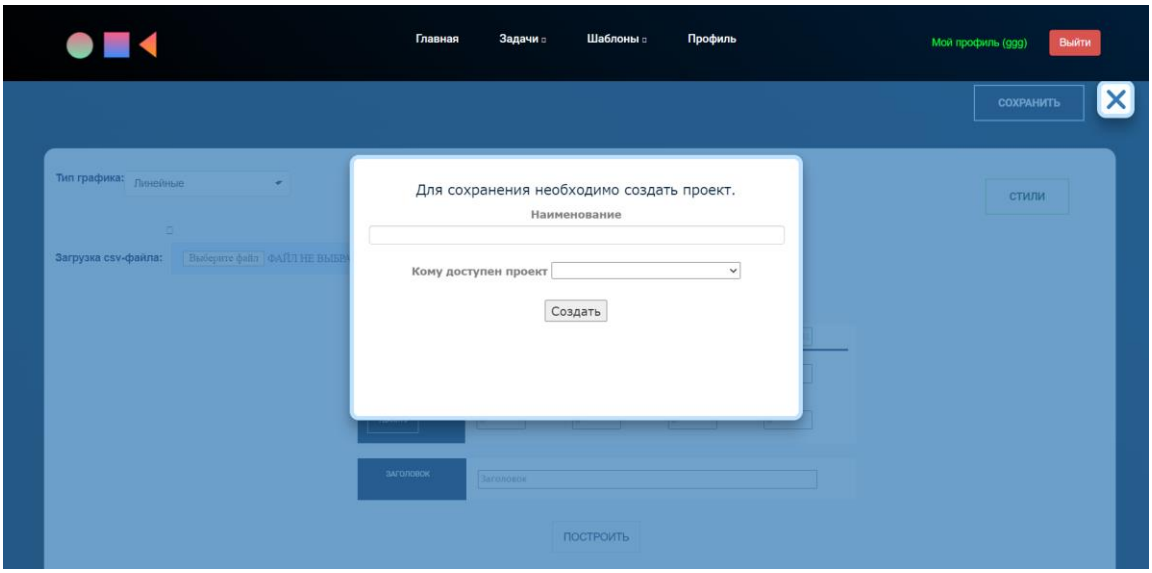

Рисунок 34 – Создание проекта

Последующее сохранение проекта сопровождается вводом комментария к сохраняемой версии проекта (рисунок 35). После сохранения проекта выводится сообщение «Изменения сохранены».

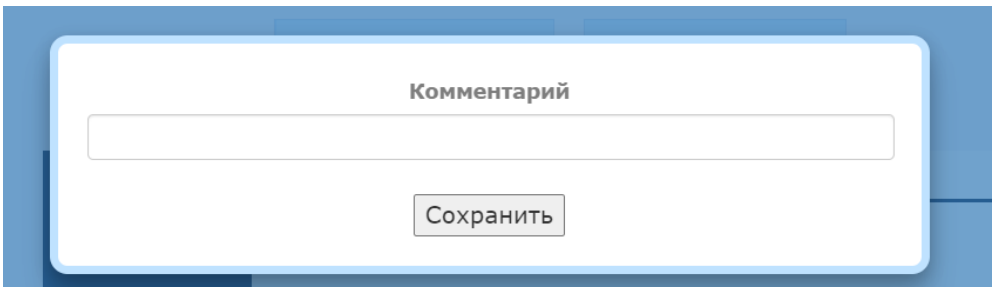

Рисунок 35 – Сохранение проекта

В профиле пользователя отображаются все проекты, к которым он имеет доступ (рисунок 36).

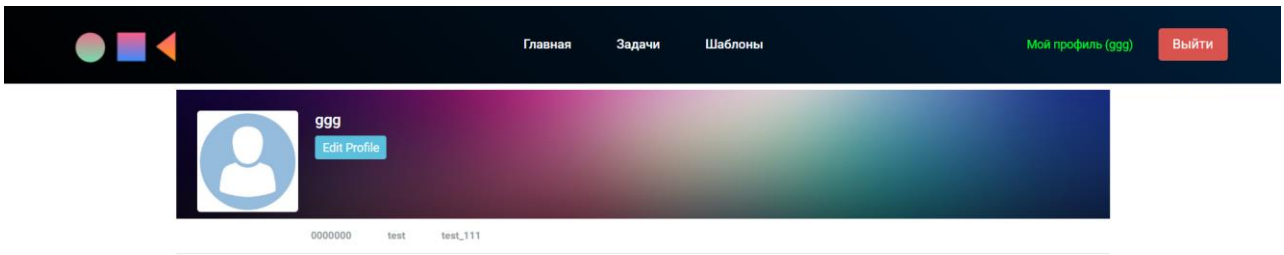

Рисунок 36 – Профиль пользователя

Список проектов представлен в виде вкладок, при нажатии на которые появляется история изменения проекта (рисунок 37). Открыть проект на момент соответствующей версии можно, нажав на кнопку «Посмотреть».

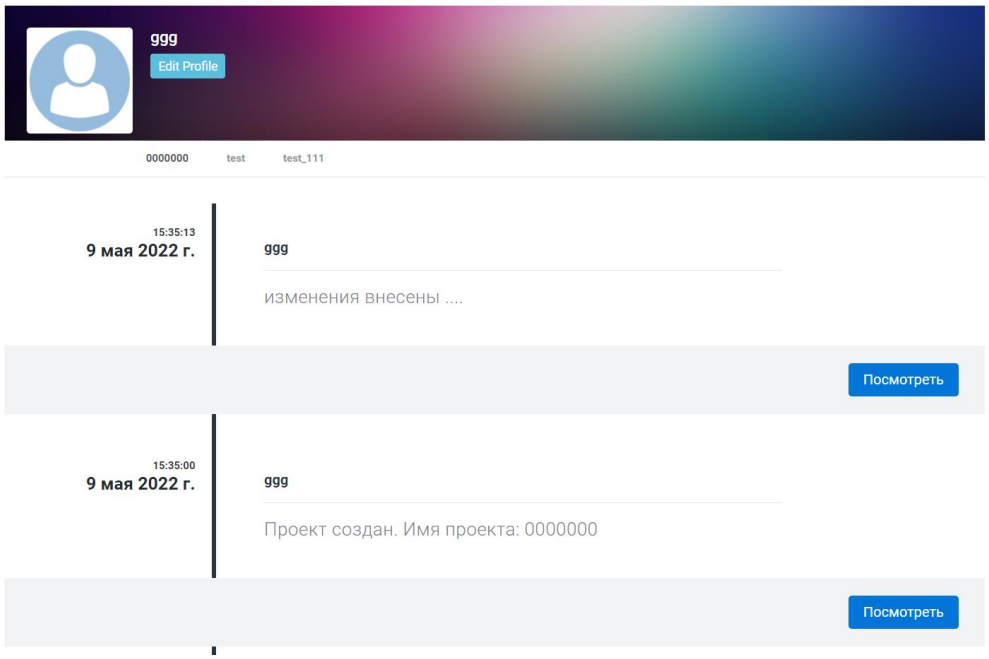

Рисунок 37 – История изменения проекта

При открытии проекта сообщение «Доступно только для просмотра.» может появится в двух случаях.

1) Если пользователь открыл не последнюю сохраненную версию проекта, а одну из предыдущих версий, то редактировать ее он не сможет (рисунок 38). Редактирование возможно только последней сохраненной версии проекта.

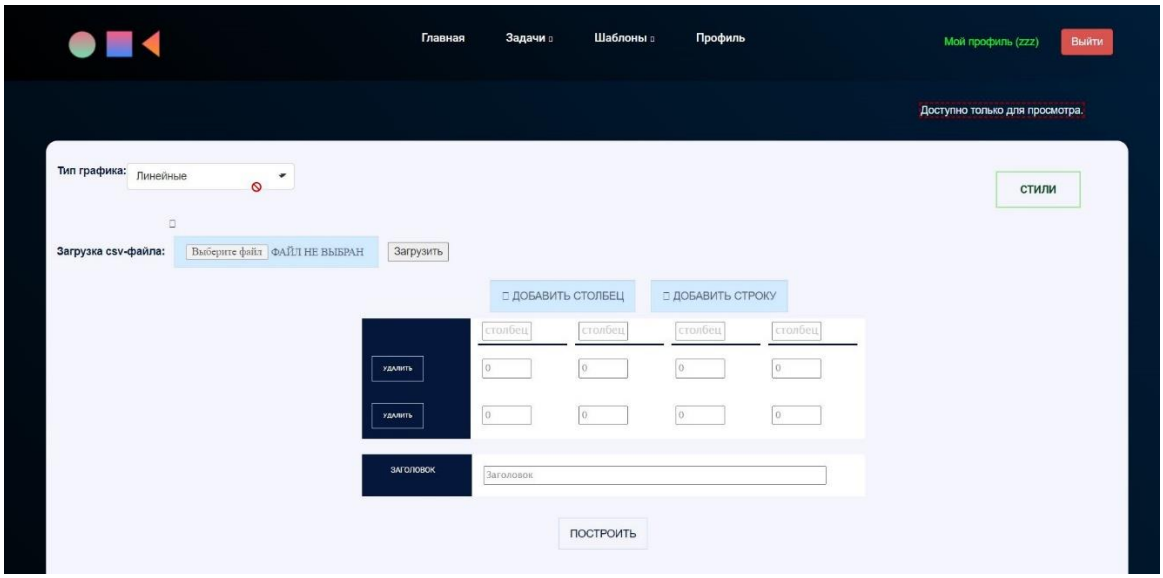

Рисунок 38 – Редактирование запрещено

2) Если одновременно над проектом работает несколько пользователей, то функция редактирования проекта доступна первому открывшему проект пользователю, остальные могут только просматривать проект. При открытии проекта в верхнем левом углу указывается активный пользователь. Если указан логин текущего пользователя, то ему доступно редактирование проекта. Иначе, будет отображаться сообщение «Доступно только для просмотра.» (рисунок 39).

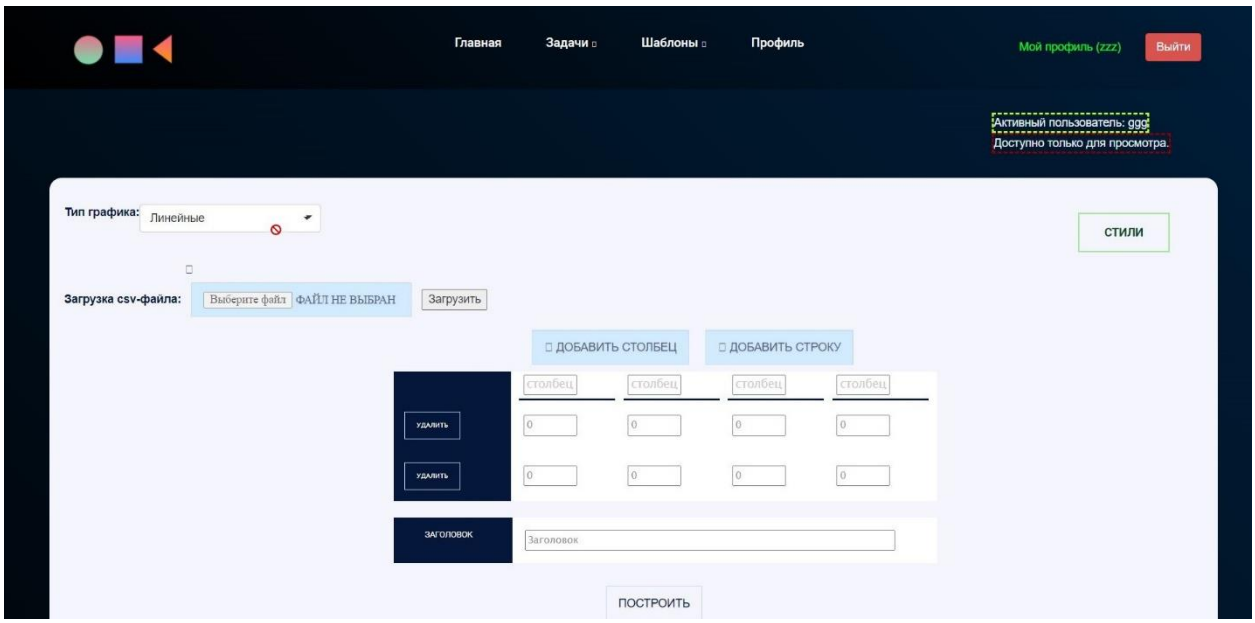

Рисунок 39 – Многопользовательская работа над проектом

ЗАКЛЮЧЕНИЕ

В итоге поэтапной реализации проекта выпускной квалификационной работы было разработано web-приложение для построения диаграмм, блок-схем, графиков онлайн. В ходе работы решены следующие задачи:

— изучены технологии и средства визуализации данных,

— выполнен анализ аналогов создаваемого продукта с выявлением их преимуществ и недостатков,

— спроектирована архитектуры приложения,

— реализована бизнес-логика приложения,

— разработан удобный пользовательский интерфейс,

— проведены отладка и тестирование продукта.

Разработанное web-приложение для онлайн построения блок-схем, диаграмм/графиков содержит функционал, способствующий упрощению работы пользователя в системе: шаблоны проекта/стиля, многопользовательская работа, экспорт/импорт данных, ведение истории версий проекта.

В приложении реализованы инструменты визуализации данных, обработки запросов клиента, соединения с базой данных и извлечения из нее информации. Интерфейс приложение построен простым и логически понятным образом, тем самым адаптирован под любую возрастную категорию пользователей.

Процесс построения блок-схемы/диаграммы/графика организован по следующим схемам: создать – сохранить или создать – выгрузить. Реализация возможности обращения пользователя к сохраненному проекту позволяет доработать проект (внести дополнительные элементы, исправить ошибку и т.п.). В свою очередь, функция совместной работы нескольких пользователей над проектом полезна для групп людей, которым необходимо решить задачу, при этом распределив обязанности.

Данное web-приложения будет очень полезно людям, которые занимаются решением задач анализа процессов, выявления зависимостей данных.

Приобретенные за период работы компетенции указаны в таблице 1.

Таблица 1 – Описание освоенных компетенций

Шифр компетенции	Расшифровка проверяемой компетенции	Расшифровка освоения компетенции
УК-1	Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач	Приобретен навык поиска информации согласно задачи, фильтрация информации по требуемым критериям
УК-2	Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений	Усовершенствован навык постановки задач и их группировки по назначению для более эффективного программирования
УК-3	Способен осуществлять социальное взаимодействие и реализовывать свою роль в команде	Усовершенствован навык эффективного распределения обязанностей
УК-4	Способен осуществлять деловую коммуникацию в устной и письменной формах на государственном языке Российской федерации и иностранном(ых) языке(ах)	Усовершенствован навык ведения диалога на тему разработки приложения со специалистами в этой области, навык грамотной постановки вопросов к научному руководителю
УК-5	Способен воспринимать межкультурное разнообразие общества в социально-историческом, этическом и философском контекстах	Толерантно и с уважением построен диалог с научным руководителем
УК-6	Способен управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни	Усовершенствован навык управления временем путем чередования задач по программирования приложения и работой над ВКР
УК-7	Способен поддерживать должный уровень физической подготовленности для обеспечения полноценной социальной и профессиональной деятельности	Приобретен навык построения работы путем чередования умственной и физической нагрузки

Продолжение таблицы 1

Шифр компетенции	Расшифровка проверяемой компетенции	Расшифровка освоения компетенции
УК-8	Способен создавать и поддерживать безопасные условия жизнедеятельности, в том числе при возникновении чрезвычайных ситуаций	Приобретен навык создания благоприятных условий для организации учебной деятельности (частые проветривания комнаты, увлажнение воздуха)
ОПК-1	Способен применять фундаментальные знания, полученные в области математических и (или) естественных наук, и использовать их в профессиональной деятельности	Применены знания в области математических наук (работа с координатами объектов) для реализации размещения блоков на рабочей области, построения графиков зависимостей данных
ОПК-2	Способен применять компьютерные/суперкомпьютерные методы, современное программное обеспечение, в том числе отечественного происхождения, для решения задач профессиональной деятельности	Усовершенствованы навыки работы с платформой ASP.NET Core MVC
ОПК-3	Способен к разработке алгоритмических и программных решений в области системного и прикладного программирования, математических, информационных и имитационных моделей, созданию информационных ресурсов глобальных сетей, образовательного контента, прикладных баз данных, тестов и средств тестирования систем и средств на соответствие стандартам и исходным требованиям	1. Приобретен навык работы с MongoDB 2. Усовершенствован навык написания программного кода на языке C# 3. Повышен уровень владения навыком работы с языком разметки HTML, таблицей стилей CSS, языком сценариев JS 4. Усовершенствован навык ручного тестирования
ОПК-4	Способен участвовать в разработке технической документации программных продуктов и комплексов с использованием стандартов, норм и правил, а также в управлении проектами создания информационных систем на стадиях жизненного цикла	Усвоен навык сопровождения проекта в рамках жизненного цикла: от написания требований к проекту до тестирования

Продолжение таблицы 1

Шифр компетенции	Расшифровка проверяемой компетенции	Расшифровка освоения компетенции
ОПК-5	Способен инсталлировать и сопровождать программное обеспечение информационных систем и баз данных, в том числе отечественного происхождения, с учетом информационной безопасности	Усовершенствован навык развертывания web-приложения в среде ASP.NET Core MVC
ПК-1	Преподавание по дополнительным общеобразовательным программам	Приобретен навык самообучения дополнительным дисциплинам, необходимым для углубленного изучения определенной задачи
ПК-2	Проверка работоспособности и рефакторинг кода программного обеспечения	Усовершенствован навык отладки программного продукта, рефакторинга кода с целью оптимизации
ПК-3	Интеграция программных модулей и компонент и верификация выпусков программного продукта	Приобретен навык разложения приложения на компоненты, построения связей между ними
ПК-4	Разработка требований и проектирование программного обеспечения	Усовершенствован навык построения web-приложения на основе модели MVC
ПК-5	Оценка и выбор варианта архитектуры программного средства	Приобретен навык анализа моделей архитектур web-приложений, выявление преимуществ и недостатков
ПК-6	Разработка тестовых случаев, проведение тестирования и исследование результатов	Усовершенствован навык критического мышления, что способствовало выявлению уязвимостей системы и нахождению решений
ПК-7	Обеспечение функционирования баз данных	Приобретен навык установки подключения к базе данных MongoDB
ПК-8	Оптимизация функционирования баз данных	Приобретен навык написания запросов к данным базы данных для получения/ обновления/ удаления записей

Продолжение таблицы 1

Шифр компетенции	Расшифровка проверяемой компетенции	Расшифровка освоения компетенции
ПК-9	Обеспечение информационной безопасности на уровне базы данных	Приобретен навык работы с библиотекой ASPNetCore.Identity, в рамках которой используется хэширования пароля пользователя
ПК-10	Выполнение работ по созданию (модификации) и сопровождению информационных систем, автоматизирующих задачи организационного управления и бизнес-процессы	Приобретен навык автоматизации бизнес-процесса с учетом всех функциональных требований и построением взаимосвязей между ними, чтобы пользователю была понятна логика приложения
ПК-11	Создание и сопровождение требований и технических заданий на разработку и модернизацию систем и подсистем малого и среднего масштаба и сложности	Приобретен навык сопровождения продукта согласно поставленным задачам разработки и модернизации приложения

СПИСОК ЛИТЕРАТУРЫ

1) Пескова О. В. О визуализации информации / О. В. Пескова // Компьютерные и информационные науки . – 2012. – № 1. – URL: https://cyberleninka.ru/article/n/o-vizualizatsii-informatsii/viewer (дата обращения: 15.05.2022).

2) Беляев Н. А. Визуализация данных : инфографика как инструмент маркетинга/ Н. А. Беляев // Маркетинг . – 2015. – № 1. – URL: https://cyberleninka.ru/article/n/vizualizatsiya-dannyh-infografika-kak -instrument marketinga/viewer (дата обращения: 01.04.2022).

3) ASP.NET Core | Полное руководство . – URL: https://metanit.com /sharp/aspnet5/ (дата обращения: 09.04.2022).

4) Общие сведения ASP.NET Core MVC | Microsoft Docs . – URL: https://docs.microsoft.com/ruru/aspnet/core/mvc/overview?view= aspnetcore-3.1 (дата обращения: 09.04.2022).

5) Шикуть А.В. Достоинства и преимущества использования C# в приложениях. / Шикуть А.В., Аристов Б.К., Просуков Е.А.// Компьютерные и информационные науки . – 2016. – № 1. – URL: https://cyberleninka.ru/article/n/osobennosti-dostoinstva-ipreimuschestvaispolzovaniya-s-v-prilozheniyah (дата обращения: 15.05.2022).

6) JavaScript documentation – DevDocs . – URL: https://devdocs.io/javascript/ (дата обращения: 13.05.2022).

7) MongoDB: The Application Data Platform | MongoDB . – URL: https:/ www.mongodb.com/ (дата обращения: 09.04.2022).

8) BSON (Binary JSON): Specification . – URL: https://bsonspec. org/spec.html (дата обращения: 09.04.2022).

9) JavaScript Canvas . – URL: https://www.javascripttutorial.net/web-apis/javascript-canvas/ (дата обращения: 18.05.2022)

10) JavaScript Prototype. – URL: https://www.javascripttutorial.net/javascript-prototype/ (дата обращения: 18.05.2022)

11) Tutorial on Creating Charts | CanvasJS JavaScript Charts . – URL: https://canvasjs.com/docs/charts/basics-of-creating-html5-chart/ (дата обращения: 18.05.2022)

12) jQuery.ajax() | jQuery API Documentation . – URL:https://api.jquery.com/jquery.ajax/ (дата обращения: 12.05.2022).

13) C# CSV - read write CSV data in C# . – URL: https://zetcode.com/csharp/csv/ (дата обращения: 14.05.2022)

ПРИЛОЖЕНИЕ

_Layout.chtml

```
// мастер-страница
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-
scale=1.0" />
<title> Online Designer </title>
<link rel="stylesheet"
href="~/lib/bootstrap/dist/css/bootstrap.min.css" />

<link rel="stylesheet" href="~/css/style.css">
</head>
<body>
<header>
<div class="header-area">
<div id="sticky-header" class="main-header-area">
<div class="container-fluid ">
<div class="header_bottom_border">
<div class="row align-items-center">
<div class="col-xl-3 col-lg-2">
<div class="logo">
<a href="index.html">
<img src="~/img/logo.png" alt="">
</a>
</div>
</div>
<div class="col-xl-6 col-lg-7">
<div class="main-menu  d-none d-lg-block">
<nav>
<ul id="navigation">
<li><a href="index.html">Главная</a></li>
<li>
<a href="#"> Задачи <i class="ti-angle-down"></i></a>
<ul class="submenu">
<li><a href="~/Project/BlockSchema"> Построение блок-схемы
</a></li>
<li><a href="~/Project/Templates"> Построение
графика/диаграммы</a></li>
</ul>
</li>
<li>
<a href="#"> Шаблоны <i class="ti-angle-down"></i></a>
<ul class="submenu">
<li><a href="~/Project/Templates"> Блок-схемы </a></li>
<li><a href="~/Project/Templates">Графики/Диаграммы
</a></li>
</ul>
</li>
```

```
</ul>
</nav>
</div>
</div>
<div class="col-xl-3 col-lg-3 d-none d-lg-block">
@{
if (User?.Identity?.IsAuthenticated ?? true)
{
<div class="Appointment">
<div class="phone_num d-none d-xl-block">
<a href="~/User/Profile" style="color:#00ff21"> Мой профиль
(@User.Identity.Name) </a>
</div>
<div class="d-none d-lg-block">
<a class="btn btn-danger" asp-controller="User" asp-
action="Logout">Выйти</a>
</div>
</div>
}
else
{
<div class="Appointment">
<div class="phone_num d-none d-xl-block">
<a href="~/User/Create">Регистрация</a>
</div>
<div class="d-none d-lg-block">
<a class="boxed-btn3" href="~/User/SignIn">Вход</a>
</div>
</div>
}
}
<div class="col-12">
<div class="mobile_menu d-block d-lg-none"></div>
</div>
</div>
</div>
</div>
</div>
</header>
<div class="modal fade" id="exampleModal" tabindex="-1"
role="dialog" aria-labelledby="exampleModalLabel" aria-
hidden="true">
<div class="modal-dialog" role="document">
<div class="modal-content">
<div class="modal-header">
<h5 class="modal-title"
id="exampleModalLabel">Регистрация</h5>
<button type="button" class="close" data-dismiss="modal"
aria-label="Close">
<span aria-hidden="true">&times;</span>
</button>
```

```
</div>
<div class="modal-body">
<form action="/Home/Create" data-ajax="true" data-ajax-
mode="replace" data-ajax-update="#results" id="form0"
method="post">
<div asp-validation-summary="ModelOnly" class="text-
danger"></div>
<div class="form-group">
<label asp-for="Name" class="control-label"></label>
<input asp-for="Name" class="form-control" />
<span asp-validation-for="Name" class="text-danger"></span>
</div>
<div class="form-group">
<label asp-for="Email" class="control-label"></label>
<input asp-for="Email" class="form-control" />
<span asp-validation-for="Email" class="text-danger"></span>
</div>
<div class="form-group">
<label asp-for="Password" class="control-label"></label>
<input asp-for="Password" class="form-control" />
<span asp-validation-for="Password" class="text-
danger"></span>
</div>
<div class="form-group">
<input type="submit" value="Зарегистрироваться" class="btn
btn-primary" />
</div>
</form>
</div>
</div>
</div>
</div>
@RenderBody()
<footer class="footer">
<div class="footer_top">
<div class="container">
<div class="row">
<div class="col-xl-3 col-md-6 col-lg-3">
<div class="footer_widget wow fadeInUp" data-wow-
duration="1s" data-wow-delay=".3s">
<div class="footer_logo">
<a href="#">
<img src="img/logo.png" alt="">
</a>
</div>
</div>
</div>
<div class="col-xl-2 col-md-6 col-lg-2">
<div class="footer_widget wow fadeInUp" data-wow-
duration="1.1s" data-wow-delay=".4s">
<h3 class="footer_title"> Главная </h3>
</div>
```

```
</div>
<div class="col-xl-3 col-md-6 col-lg-3">
<div class="footer_widget wow fadeInUp" data-wow-
duration="1.2s" data-wow-delay=".5s">
<h3 class="footer_title"> Задачи </h3>
<ul>
<li><a href="candidate.html"> Построение блок-схемы
</a></li>
<li><a href="elements.html"> Построение
графика/диаграммы</a></li>
</ul>
</div>
</div>
<div class="col-xl-4 col-md-6 col-lg-4">
<div class="footer_widget wow fadeInUp" data-wow-
duration="1.3s" data-wow-delay=".6s">
<h3 class="footer_title">
Шаблоны
</h3>
<ul class="submenu">
<li><a href="blog.html"> Блок-схемы </a></li>
<li><a href="single-blog.html">Графики/Диаграммы </a></li>
</ul>
</div>
</div>
</div>
</div>
</div>
<div class="copy-right_text wow fadeInUp" data-wow-
duration="1.4s" data-wow-delay=".3s">
<div class="container">
<div class="footer_border"></div>
<div class="row">
<div class="col-xl-12">
<p class="copy_right text-center">
Online Designer &copy;
<script>document.write(new Date().getFullYear());</script>
</p>
</div>
</div>
</div>
</div>
</footer>
<script src="/lib/jquery/dist/jquery.min.js"></script>
<script
src="/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></scrip
t>
<script src="/js/site.js" asp-append-
version="true"></script>
@await RenderSectionAsync("Scripts", required: false)
</body>
</html>
```

UserController.cs

```
// контроллер обработки событий регистрации, авторизации пользователя, выхода из системы, получения списка проектов пользователя
public class UserController : Controller
{
// инициализация объектов для работы с авторизованным пользователем
private UserManager<AppUser> _userManager;
private SignInManager<AppUser> _signInManager;
public UserController(UserManager<AppUser> userManager, SignInManager<AppUser> signInManager)
{
this._userManager = userManager;
this._signInManager = signInManager;
}

public IActionResult Create()
{
return View();
}

// асинхронный метод, который возвращает список доступных пользователю проектов, также используется для ajax-запроса по отображению версии проекта
public async Task<IActionResult> ProfileAsync(string name)
{
// подключение к коллекции БД
string connectionString = "mongodb://localhost";
var client = new MongoClient(connectionString);
var database = client.GetDatabase("IdentityAuthDB");
var collection = database.GetCollection<HistoryProject>("HistoryProjects");
var ProjectCollection = database.GetCollection<Project>("Project");
// если есть имя проекта
if (!String.IsNullOrEmpty(projectName))
{
// фильтр по наименованию, дате и времени создания проекта
var filter = new BsonDocument("$or", new BsonArray{
new BsonDocument("Name", projectName), new BsonDocument("timeCreate", time), new BsonDocument("dateCreate", date)});

var projects = await collection.Find(filter).ToListAsync();

string data = projects[0].data.ToString();
```

```
string chartNumber = projects[0].typeChart;
string titleDiag = projects[0].titleDiag;
// перенаправление на страницу проекта
return RedirectToAction("Project", "Project", new { data =
                            data, chartNumber = chartNumber,
                            titleDiag = titleDiag});
}
// если есть имя проекта, т.е. это ситуация, когда
                            формируется список доступных
                            проектов
if (!String.IsNullOrEmpty(name))
{
// поиск, есть ли проекты, где автор проекта пользователь
                            или ему доступен проекта
var filter_1 = new BsonDocument("$or", new BsonArray{
new BsonDocument("availableToWhom", name), new
                            BsonDocument("AuthLogin",
                            name)});

var proj = await
                            ProjectCollection.Find(filter_1).
                            ToListAsync();
// формирование списка наименований проектов, доступных
                            пользователю
List<string> list_proj_name = new List<string>();

foreach (var el in proj)
{
list_proj_name.Add(el.Name);
}
// фильтр поиска записей об изменении проекта в БД по
                            наименованию
var filter = new BsonDocument("$or", new BsonArray{
new BsonDocument("Name", list_proj_name[0])});

var projects = await collection.Find(filter).ToListAsync();
// формирование списка проектов для передачи в представление
for (int i = 1; i < list_proj_name.Count; i++)
{
filter = new BsonDocument("$or", new BsonArray{
new BsonDocument("Name", list_proj_name[i])});
var temp = await collection.Find(filter).ToListAsync(); ;
projects.Add(temp[0]);
}

return View(projects);
}

return View();
}

public IActionResult SignIn(string message)
```

```
{
if (!String.IsNullOrEmpty(message))
ViewBag.Message = message;
return View();
}

// метод, отвечающий за регистрацию пользователя в системе
[HttpPost]
public async Task<IActionResult> Create(User user)
{
if (ModelState.IsValid)
{
AppUser appUser = new AppUser
{
UserName = user.Name,
Email = user.Email
};

IdentityResult result = await
                        _userManager.CreateAsync(appUser,
                        user.Password);

if (result.Succeeded)
{
return RedirectToAction("SignIn", "User", new { message =
                        "Пользователь успешно
                        зарегистрирован." });
}
else
{
foreach (IdentityError error in result.Errors)
ModelState.AddModelError("", error.Description);
return View();
}
}
return View();
}

// метод, отвечающий за авторизацию пользователя в системе
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<IActionResult> SignIn([Required] string
                        name, [Required] string password,
                        string returnURL)
{
if (ModelState.IsValid)
{
AppUser appUser = await _userManager.FindByNameAsync(name);

if (appUser != null)
{
```

```
Microsoft.AspNetCore.Identity.SignInResult result = await
                            _signInManager.PasswordSignInAsyn
                            c(appUser,    password,    false,
                            false);

if (result.Succeeded)
{
return RedirectToAction("Profile", "User");
}
}

ModelState.AddModelError(nameof(name),    "Login    Failed:
                            Invalid Name or Password.");
}

return View();
}
// метод, отвечающий за выход пользователя из системы
[Authorize]
public async Task<IActionResult> Logout()
{
await _signInManager.SignOutAsync();
return RedirectToAction("Index", "Home");
}
}
```

Project.cs

```
// модель записи о создании проекта

public class Project
    {
        public string Name { get; set;}

        public string AuthLogin { get; set;}

        public List<string> availableToWhom { get; set; }
    }
```

HistoryProject.cs

```
// модель записи об внесении изменений проекта

[BsonIgnoreExtraElements]
    public class HistoryProject
    {
        public string Name { get; set;}
        public string AuthLogin { get; set; }
        public string redactorLogin { get; set;}
        public List<List<DataPoint_1>> data { get; set;}
        public string typeChart { get; set;}
        public List<List<string>> style { get; set;}
        [Required]
        public String timeCreate { get; set;}
        public String dayCreate { get; set; }
```

```
        public List<string> comment { get; set; }
    }
```

DataPoint_1.cs

```
// модель точки графика
[DataContract]
public class DataPoint_1
{
     public DataPoint_1(string label, double y)
     {
          this.Label = label;
          this.Y = y;
     }

     [DataMember(Name = "label")]
     public string Label = "";

     [DataMember(Name = "y")]
     public Nullable<double> Y = null;
     }
```

DataPoint_2.cs

```
// модель точки графика
[DataContract]
public class DataPoint_2
{
public DataPoint_2(string label, double x, double y, double
     z)
{
this.Label = label;
this.X = x;
this.Y = y;
this.Z = z;
}


[DataMember(Name = "label")]
public string Label = "";

[DataMember(Name = "x")]
public Nullable<double> X = null;

[DataMember(Name = "y")]
public Nullable<double> Y = null;

[DataMember(Name = "z")]
public Nullable<double> Z = null;
}
```

DataPoint_3.cs

```
// модель точки графика
[DataContract]
public class DataPoint_3
{
public DataPoint_3(DateTime x, double[] y)
{
this.X = x;
this.Y = y;
}
[DataMember(Name = "x")]
public DateTime X;

[DataMember(Name = "y")]
public double[] Y = null;
}
```

ProjectController.cs

```
// контроллер обработки событий сохранения, создания проекта, загрузки проекта
// асинхронный метод для создания и сохранения проекта, работа с данными графика
public async Task<ActionResult> Project(string Status, string Name, string chartNumber, string data, string comment, string availableToWhom, string titleDiag)
{
// подключение к коллекциям БД
string connectionString = "mongodb://localhost";
var client = new MongoClient(connectionString);
var database = client.GetDatabase("IdentityAuthDB");
var HistoryProjects = database.GetCollection<HistoryProject>("HistoryProjects");
var Projects = database.GetCollection<Project>("Projects");

// если данные введены
if (!String.IsNullOrEmpty(data))
{
// преобразование строки данных
string[] splitString = data.Split('/');
string[] title = splitString[0].Split(',');

// определение типа точки по типу диаграммы
var point_type = "";
```

```
foreach(var element in charts_type)
{
if (element.Key.Equals(chartNumber.ToString()))
{
point_type = element.Value;
}
}

if (point_type == "DataPoint_1")
{
ViewBag.DataPoints =
                          JsonConvert.SerializeObject(Data_
                          001(splitString, title));
}

if (point_type == "DataPoint_2")
{
ViewBag.DataPoints =
                          JsonConvert.SerializeObject(Data_
                          002(splitString, title));
}

if (point_type == "DataPoint_3")
{
ViewBag.DataPoints =
                          JsonConvert.SerializeObject(Data_
                          003(splitString, title));
}

ViewBag.Chart = JsonConvert.SerializeObject(chartNumber);

if (!String.IsNullOrEmpty(Name))
{

if (!String.IsNullOrEmpty(Status))
{
// создание документа об изменении проекта
HistoryProject projectSave = new HistoryProject();
projectSave.Name = Name;
projectSave.redactorLogin = User.Identity.Name;
projectSave.data = data_points;
projectSave.titleDiag = titleDiag;
projectSave.dayCreate = DateTime.Now.ToLongDateString();
projectSave.timeCreate = DateTime.Now.ToLongTimeString();
projectSave.typeChart = chartNumber;
projectSave.comment = new List<string>() {
                          User.Identity.Name, comment };

await HistoryProjects.InsertOneAsync(projectSave);

return PartialView("Project_Saved");
```

```
}

// создание записи о создании проекта в истории проекта
HistoryProject project = new HistoryProject();
project.Name = Name;
project.redactorLogin = User.Identity.Name;
project.data = data_points;
projectSave.titleDiag = titleDiag;
project.dayCreate = DateTime.Now.ToLongDateString();
project.timeCreate = DateTime.Now.ToLongTimeString();
project.typeChart = chartNumber;
project.comment = new List<string>() { User.Identity.Name,
                          "Проект создан. Имя проекта: " +
                          Name.ToString()};
// создания документа о проекте
Project proj = new Project();
proj.Name = Name;
proj.AuthLogin = User.Identity.Name;
List<string> usersLogin = new List<string>();
usersLogin.Add(availableToWhom);
proj.availableToWhom=usersLogin;

await HistoryProjects.InsertOneAsync(project);
await Projects.InsertOneAsync(proj);

return PartialView("Project_Created");
}

return PartialView(chartNumber, data);
}
return View();
}
// метод для приведения данных к точкам первого типа
public List<List<DataPoint_1>> Data_001(string[]
                          splitString, string[] title)
{
List<List<DataPoint_1>> data_points = new
                          List<List<DataPoint_1>>();

for (int i = 1; i < splitString.Length - 1; i++)
{
List<DataPoint_1> points = new List<DataPoint_1>();

string[] numbers = splitString[i].Split(',');

for (int j = 1; j < numbers.Length - 1; j++)
{
points.Add(new DataPoint_1(title[j - 1],
                          Convert.ToDouble(numbers[j])));
}

data_points.Add(points);
```

```
}

return data_points;
}
// метод для приведения данных к точкам второго типа
public List<List<DataPoint_2>> Data_002(string[]
                            splitString, string[] title)
{
List<List<DataPoint_2>> data_points = new
                            List<List<DataPoint_2>>();

for (int i = 1; i < splitString.Length - 1; i++)
{
List<DataPoint_2> points = new List<DataPoint_2>();

string[] numbers = splitString[i].Split(',');

for (int j = 1; j < numbers.Length - 1; j+=3)
{
points.Add(new DataPoint_2(title[j - 1],
                            Convert.ToDouble(numbers[j]),
                            Convert.ToDouble(numbers[j+1]),
                            Convert.ToDouble(numbers[j+2])));
}

data_points.Add(points);
}

return data_points;
}
// метод для приведения данных к точкам третьего типа
public List<List<DataPoint_3>> Data_003(string[]
                            splitString, string[] title)
{
List<List<DataPoint_3>> data_points = new
                            List<List<DataPoint_3>>();

for (int i = 1; i < splitString.Length - 1; i++)
{
List<DataPoint_3> points = new List<DataPoint_3>();

string[] numbers = splitString[i].Split(',');

for (int j = 1; j < numbers.Length - 1; j+=4)
{
points.Add(new DataPoint_3(DateTime.ParseExact(title[j - 1],
                            new string[] { "MM.dd.yyyy", "MM-
                            dd-yyyy", "MM/dd/yyyy" },
                            provider, DateTimeStyles.none),
                            new double[] {
                            Convert.ToDouble(numbers[j]),
                            Convert.ToDouble(numbers[j+1]),
```

```
                                    Convert.ToDouble(numbers[j+2]),
                                    Convert.ToDouble(numbers[j+3])
                                    }));
    }

    data_points.Add(points);
    }

    return data_points;
    }
```

Project_Diag.chtml

```
    // интерфейс страницы проекта диаграммы с JavaScript
    <div class="whereTable">

    <div style="padding-left:160px;">
    <button type="button" id="btnAddCol" class="button">
    <i class="fa fa-credit-card"></i>
    <span> Добавить столбец </span>
    </button>

    <button type="button" id="btnAddRow" class="button">
    <i class="fa fa-credit-card"></i>
    <span> Добавить строку </span>
    </button>
    </div>

    <table id="myform" class="table_col">
    <colgroup>
    <col style="background:#05173a;">
    </colgroup>
    <tr>
    <th> </th>
    <th      scope="col"><input      type="text"      id="cb_01"
placeholder='столбец' /></th>
    <th      scope="col"><input      type="text"      id="cb_02"
placeholder='столбец' /></th>
    <th      scope="col"><input      type="text"      id="cb_03"
placeholder='столбец' /></th>
    <th      scope="col"><input      type="text"      id="cb_04"
placeholder='столбец' /></th>
    </tr>
    <tr>
    <td>       <input       type='button'       value='Удалить'
onclick='removeRow(this,1);'  style='position:  relative;  border:
1px solid #C7DAF0; outline: none; display: inline-block; overflow:
hidden; text-align: center; text-decoration: none; text-transform:
uppercase; cursor: pointer; font-size: 8px; padding: 10px 14px;
color: white; background: transparent; text-shadow: 0 1px 1px
rgba(0, 0, 0, 0.2); margin: 8px;' /></td>
    <td><input type="text" id="cb_11" value="0" /></td>
```

```
<td><input type="text" id="cb_12" value="0" /></td>
<td><input type="text" id="cb_13" value="0" /></td>
<td><input type="text" id="cb_14" value="0" /></td>
</tr>
<tr>
<td>       <input       type='button'       value='Удалить'
onclick='removeRow(this,2);'  style='position:  relative;  border:
1px solid #C7DAF0; outline: none; display: inline-block; overflow:
hidden; text-align: center; text-decoration: none; text-transform:
uppercase; cursor: pointer; font-size: 8px; padding: 10px 14px;
color: white; background: transparent; text-shadow: 0 1px 1px
rgba(0, 0, 0, 0.2); margin: 8px;' /></td>
<td><input type="text" id="cb_21" value="0" /></td>
<td><input type="text" id="cb_22" value="0" /></td>
<td><input type="text" id="cb_23" value="0" /></td>
<td><input type="text" id="cb_24" value="0" /></td>
</tr>
</table>
<br>
<table id="myform_1" class="table_col">
<colgroup>
<col style="background:#05173a;">
</colgroup>
<tr>
<td> <p style='position: relative; outline: none; display:
inline-block;  overflow:  hidden;  text-align:  center;  text-
decoration: none; text-transform: uppercase; cursor: pointer;
font-size: 10px; color: white; background: transparent; text-
shadow: 0 1px 1px rgba(0, 0, 0, 0.2); padding-left: 33px; padding-
right: 33px;'> Заголовок </p></td>
<td><input    type="text"    id="ttl"    value="Заголовок"
style="width: 450px;" /></td>
</tr>
</table>
<br>
<div style="padding-left:250px;">
<input    id="submit"    type="submit"    value="Построить"
style="position: relative; border: 1px solid #C7DAF0; outline:
none; display: inline-block; overflow: hidden; text-align: center;
text-decoration: none; text-transform: uppercase; cursor: pointer;
font-size: 14px; padding: 10px 14px; color: #05173a; background:
transparent; text-shadow: 0 1px 1px rgba(0, 0, 0, 0.2); margin:
8px;">
</div>
</div>

<!-- Модальное окно -->
<input type="checkbox" id="css-modal-checkbox" />
<div class="cmc">
<div class="cmt">
<div id="mess">
@{
```

```
     if (User?.Identity?.IsAuthenticated ?? true)
     {
     <section id="message">
     </section>
     <div id="content_1">
     <h4 style="color:#001D38">Для сохранения необходимо создать
проект.</h4>
     <label class="control-label"> Наименование </label>
     <input  class="form-control"  type="text"  id="projectName"
style="width: 550px;" />
     <br>
     <label class="control-label"> Кому доступен проект </label>
     <select       id="availableToWhom"       class="js-select2"
name="usersLogin"      placeholder="Выберите      пользователей"
style="width: 250px;">
     </select>

     <link                                          rel="stylesheet"
href="/select2/dist/css/select2.min.css">
     <script src="/jquery.min.js"></script>
     <script src="/select2/dist/js/select2.min.js"></script>
     <script src="/select2/dist/js/i18n/ru.js"></script>

     <br>
     <br>
     <input id="create" type="submit" value="Создать">
     </div>

     <div id="content_2" style="opacity:0;">
     <label class="control-label"> Комментарий </label>
     <input   class="form-control"   type="text"   id="comment"
style="width: 550px;" />
     <br>
     <input id="saveComment" type="submit" value="Сохранить">
     </div>
     }
     else
     {
     <div id="title">
     <h3>Для сохранения проекта необходимо авторизоваться.</h3>

     <div class="d-none d-lg-block">
     <a class="boxed-btn3" id="signIn">Вход</a>
     </div>
     </div>

     <div class="wrapper" id="wrapper_1" style="opacity:0;">
     <input   type="submit"   value="Регистрация"   class="btn"
id="btn_1"   style="color:#2D6B9F;   font-weight:500;   margin-
left:400px;" />

     <h4 class="sign-in">Войти</h4>
```

```
    <div class="clear"> @Html.Raw(ViewBag.Message) </div>

    <form asp-controller="User" asp-action="SignIn">
    <div     asp-validation-summary="ModelOnly"     class="text-
danger"></div>
    <div class="form-group">
    <label asp-for="Name" class="control-label"></label>
    <input   asp-for="Name"   class="form-control"   type="text"
style="width:550px;" />
    <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <div class="form-group">
    <label asp-for="Password" class="control-label"></label>
    <input   asp-for="Password"   class="form-control"   value=""
type="password" style="width:550px;" />
    <span       asp-validation-for="Password"       class="text-
danger"></span>
    </div>
    <div class="form-group">
    <input type="submit" value="Войти" class="btn btn-primary" />
    </div>
    </form>
    </div>
    <div class="wrapper" id="wrapper_2">
    <div>
    <input  type="submit"  value="Войти"  class="btn"  id="btn_2"
style="color:#2D6B9F; font-weight:500; margin-left:400px;" />
    <h4 class="sign-in">Регистрация</h4>
    </div>
    <div class="clear"></div>

    <form asp-action="Create">
    <div     asp-validation-summary="ModelOnly"     class="text-
danger"></div>
    <div class="form-group">
    <label asp-for="Name" class="control-label"></label>
    <input   asp-for="Name"   class="form-control"   type="text"
style="width:550px;" />
    <span asp-validation-for="Name" class="text-danger"></span>
    </div>
    <div class="form-group">
    <label asp-for="Email" class="control-label"></label>
    <input  asp-for="Email"  class="form-control"  type="email"
style="width:550px;" />
    <span asp-validation-for="Email" class="text-danger"></span>
    </div>
    <div class="form-group">
    <label asp-for="Password" class="control-label"></label>
    <input        asp-for="Password"        class="form-control"
type="password" style="width:550px;" />
    <span       asp-validation-for="Password"       class="text-
danger"></span>
```

```
     </div>
     <div class="form-group">
     <input type="submit" value="Зарегистрировать" class="btn btn-
primary" />
     </div>
     </form>
     </div>
     <script>
     $("#wrapper_1").delay(0).slideUp(0);
     $("#wrapper_2").delay(0).slideUp(0);
     </script>
     }
     }
     </div>
     </div>
     <label      for="css-modal-checkbox"      class="css-modal-
close"></label>
     </div>
     </div>

     <div class="row">
     <div id="chartContainer" style="height: 370px; width: 100%;
margin-top:25px;"></div>
     <div style="padding-left:1200px;">
     <button type="button" id="savePNG" class="button">
     <i class="fa fa-credit-card"></i>
     <span> Сохранить как картинку </span>
     </button>
     </div>
     </div>
     <script type="text/javascript">
     var myform = $('#myform'),
     projectName = "",
     iterRow = 0,
     col = 5,
     row = 2;

     // добавление столбцов
     $('#btnAddCol').click(function () {
     myform.find('tr').each(function () {
     var trow = $(this);
     if (trow.index() === 0) {
     trow.append("<th><input     type='text'     id='cb_0"     +
col.toString() + "' placeholder='столбец'/></th>");
     col += 1;
     } else {
     trow.append("<td><input     type='text'     id='cb_"     +
trow.index().toString()  +  ""  +  (col  -  1).toString()  +  "'
value='0'/></td>");
     }
     });
     iterRow += 1;
```

```
    });

    // добавление строк
    $('#btnAddRow').click(function () {
    row += 1;
    var newRow = document.all("myform").insertRow();
    var tab = document.all("myform");
    var oCell = newRow.insertCell();
    oCell.innerHTML = "<td> <input type='button' value='Удалить'
onclick='removeRow(this," + row + ");' style='position: relative;
border: 1px solid #C7DAF0; outline: none; display: inline-block;
overflow: hidden; text-align: center; text-decoration: none; text-
transform: uppercase; cursor: pointer; font-size: 8px; padding:
10px 14px; color: white; background: transparent; text-shadow: 0
1px 1px rgba(0, 0, 0, 0.2); margin: 8px;'/></td>";
    for (let i = 1; i < col; i++) {
    oCell = newRow.insertCell();
    oCell.innerHTML = "<td><input type='text' id='cb_" +
row.toString() + "" + i.toString() + "' value='0'/></td>";
    }
    console.log("Новая строка");
    });


    // удаление строк
    function removeRow(element, num) {
    document.all("myform").deleteRow(num);
    row -= 1;
    }

    // вызов ajax-запроса для построения диаграммы
    $('#submit').click(function () {
    var data = [];

    for (let i = 0; i < row+1; i++) {
    var rowData = [];
    for (let j = 1; j < col; j++) {
    rowData.push(document.getElementById("cb_" + i.toString() +
"" + j.toString()).value);
    }

    rowData.push("/");
    data.push(rowData);
    }
    var e = document.getElementById("list");
    var indexValue = e.options[e.selectedIndex].value;
    var title = document.getElementById("ttl").value;
    console.log(title);
    $('#chartContainer').load('@Url.Action("Project",
"Project")?chartNumber=' + indexValue + '&data=' + data +
'&titleDiag=' + title)
    console.log(title);
```

```
    });

    // вызов ajax-запроса для создания проекта
    $('#create').click(function () {
    var data = [];
    console.log("Зашел 1");
    for (let i = 0; i < row+1; i++) {
    var rowData = [];
    for (let j = 1; j < col; j++) {
    rowData.push(document.getElementById("cb_" + i.toString() +
"" + j.toString()).value);
    }
    rowData.push("/");
    data.push(rowData);
    }
    var e = document.getElementById("list");
    var indexValue = e.options[e.selectedIndex].value;
    var users = document.getElementById("availableToWhom");
    var Value = users.options[e.selectedIndex].value;
    var name = document.getElementById("projectName").value;
    projectName = name;
    $('#message').load('@Url.Action("Project", "Project")?Name='
+ name + '&chartNumber=' + indexValue + '&data=' + data + +
'&availableToWhom=' + Value);

    $("#content_2").delay(3000).animate({ 'opacity': '1' }, 500);
    $("#content_1").delay(0).slideUp(0);
    });

    // вызов ajax-запроса для сохранения проекта
    $('#saveComment').click(function () {
    var data = [];
    for (let i = 0; i < row+1; i++) {
    var rowData = [];
    for (let j = 1; j < col; j++) {
    rowData.push(document.getElementById("cb_" + i.toString() +
"" + j.toString()).value);
    }
    rowData.push("/");
    data.push(rowData);
    }
    var e = document.getElementById("list");
    var indexValue = e.options[e.selectedIndex].value;
    var comment = document.getElementById("comment").value;
    $('#message').load('@Url.Action("Project",
"Project")?Status="Save"&Name=' + projectName + '&chartNumber=' +
indexValue + '&data=' + data + '&comment=' + comment);

    // обработка появлений и исчезновений надписей/кнопок
    $("#message").delay(0).slideDown(0);
    $("#message").delay(2500).slideUp(300);
    });
```

```
    $('#signIn').click(function () {
    $("#wrapper_1").delay(0).slideDown(0);
    $("#wrapper_1").delay(0).animate({ 'opacity': '1' }, 0);
    $("#title").delay(0).slideUp(0);
    });

    $('#btn_1').click(function () {
    $("#wrapper_1").delay(0).slideUp(0);
    $("#wrapper_2").delay(0).slideDown(0);
    });
    $('#btn_2').click(function () {
    $("#wrapper_2").delay(0).slideUp(0);
    $("#wrapper_1").delay(0).slideDown(0);
    });

// выгрузка диаграммы в виде картинки
document.getElementById("savePNG").addEventListener("click",
function () {
html2canvas(document.getElementById("chartContainer"),
    {
    allowTaint: true,
    useCORS: true
    }).then(function (canvas) {
    var anchorTag = document.createElement("a");
    document.body.appendChild(anchorTag);
    anchorTag.download = "image.jpg";
    anchorTag.href = canvas.toDataURL();
    anchorTag.target = '_blank';
    anchorTag.click();
    });
    });
    </script>
```

Project_BS.chtml

```
    // интерфейс страницы проекта блок-схемы с JavaScript
    <div class="content-wrapper" style="margin-top: 30px; margin-
left:40px; margin-right:30px;">

    <div class="content-body">
    <div class="content ">
    <div class="row">
    <canvas id="canvas"></canvas>
    <input id="input" type="file" accept=".json">
    <input id="source-input" type="file" accept=".cpp, .hpp, .c,
.h, .pas">
    <input id="text-input" type="text">
    </div>
    </div>
    </div>
    </div>
```

```
    <script>
    // описание прототипов функции точки соединения
function Connector(block, dx, dy) {
this.block = block
this.dx = dx
this.dy = dy
this.Update()
}

Connector.prototype.Draw = function (ctx, x0, y0, scale, isActive
= true) {
let x = this.posX * scale + x0
let y = this.posY * scale + y0
ctx.lineWidth = BLOCK_LINE_WIDTH
ctx.fillStyle = BLOCK_CONNECTOR_COLOR[DARK_THEME]
ctx.beginPath()
ctx.arc(x, y, CONNECTION_RADIUS * scale, 0, 2 * Math.PI)
ctx.fill()
ctx.strokeStyle = isActive ? ACTIVE_CONNECTOR_COLOR[DARK_THEME] :
HOVER_CONNECTOR_COLOR[DARK_THEME]
ctx.beginPath()
ctx.arc(x, y, (CONNECTION_RADIUS + 2) * scale, 0, 2 * Math.PI)
ctx.stroke()
}

Connector.prototype.GetDistance = function (x, y) {
let dx = x - this.posX
let dy = y - this.posY
return dx * dx + dy * dy
}

Connector.prototype.IsMouseHover = function (x, y) { return
this.GetDistance(x, y) < MOUSE_STEP * MOUSE_STEP }

Connector.prototype.Update = function () {
this.x = this.block.width * this.dx
this.y = this.block.height * this.dy
this.posX = this.block.x + this.x + Math.sign(this.dx) * GRID_SIZE
this.posY = this.block.y + this.y + Math.sign(this.dy) * GRID_SIZE
}

// описание прототипов функции блока
function Block(x, y, text, width, height, type = BLOCK_TYPE,
isMenuBlock = false) {
this.x = x
this.y = y
this.field = new TextField(text == '' && !isMenuBlock ?
this.GetDefaultText(type) : text)
if (width < RESIZE_DISTANCE || height < RESIZE_DISTANCE) throw
"invalid block size"
this.width = width
this.height = height
```

```
if (ALL_BLOCK_TYPES.indexOf(type) == -1) throw "invalid block
type"
this.type = type
this.isMenuBlock = isMenuBlock
this.fontSize = BLOCK_FONT_SIZE
this.textHeight = BLOCK_TEXT_HEIGHT
this.textWidth = BLOCK_TEXT_WIDTH
this.isBold = false
this.isItalic = false
this.textAlign = CENTER_TEXT_ALIGN
if (type == TEXT_TYPE) this.textAlign = LEFT_TEXT_ALIGN
this.labelsPosition = isMenuBlock ? 0 : 1
this.FixPositionByGrid()
this.InitResizePoints()
this.InitConnectors()
this.FixHeightByText()
}

Block.prototype.GetDefaultText = function (type) {
if (type == BEGIN_END_TYPE) return 'начало'
if (type == IN_OUT_TYPE || type == DISPLAY_TYPE) return 'вывод'
if (type == FOR_LOOP_TYPE || type == FOR_LOOP_BEGIN_TYPE) return
'i от 1 до n'
if (type == LABEL_TYPE) return '1'
if (type == TEXT_TYPE) return 'change text'
return ''
}

Block.prototype.FixHeightByText = function () {
let totalHeight = this.field.texts.length * this.textHeight
if (totalHeight > this.height || this.type == TEXT_TYPE) {
let newHeight = Math.floor((totalHeight + GRID_SIZE * 2 - 1) /
GRID_SIZE / 2) * GRID_SIZE * 2
this.y += (newHeight - this.height) / 2
this.height = newHeight
} this.FixPoints()
}

Block.prototype.FixWidthByText = function (ctx) {
ctx.font = this.GetFormatting() + (this.fontSize) + 'px ' +
BLOCK_FONT
let maxWidth = this.GetMaxWidth(ctx)
let currWidth = this.width
this.width = Math.floor(maxWidth / GRID_SIZE + 1) * GRID_SIZE
if (this.textAlign == LEFT_TEXT_ALIGN) this.x += (this.width -
currWidth) / 2
this.FixPoints()
}
Block.prototype.FixSizesByText = function (ctx) {
ctx.font = this.GetFormatting() + (this.fontSize) + 'px ' +
BLOCK_FONT
this.FixHeightByText()
```

```
if (this.type == TEXT_TYPE) this.FixWidthByText(ctx)
}
Block.prototype.InitBorders = function () {
this.left = this.x - this.width / 2
this.right = this.x + this.width / 2
this.top = this.y - this.height / 2
this.bottom = this.y + this.height / 2
}

Block.prototype.FixPositionByGrid = function () {
this.x = Math.floor((this.x + GRID_SIZE - 1) / GRID_SIZE) *
GRID_SIZE
this.y = Math.floor((this.y + GRID_SIZE - 1) / GRID_SIZE) *
GRID_SIZE
this.height = Math.floor((this.height + GRID_SIZE - 1) /
GRID_SIZE) * GRID_SIZE
this.width = Math.floor((this.width + GRID_SIZE - 1) / GRID_SIZE)
* GRID_SIZE
this.InitBorders()
}
Block.prototype.InitConnectors = function () {
this.connectors = []
this.connectors[TOP_CONNECTOR] = new Connector(this, 0, -0.5)
this.connectors[RIGHT_CONNECTOR] = new Connector(this, 0.5, 0)
this.connectors[BOTTOM_CONNECTOR] = new Connector(this, 0, 0.5)
this.connectors[LEFT_CONNECTOR] = new Connector(this, -0.5, 0)
}
Block.prototype.UpdateConnectors = function () { for (let i = 0;
i < this.connectors.length; i++)this.connectors[i].Update() }
Block.prototype.GetFormatting = function () {
let bold = this.isBold ? 'bold ' : ''
let italic = this.isItalic ? 'italic ' : ''
return bold + italic
}
Block.prototype.DrawBlock = function (ctx, x0, y0, scale) {
ctx.beginPath()
ctx.rect(this.left * scale + x0, this.top * scale + y0, this.width
* scale, this.height * scale)
ctx.stroke()
ctx.fill()
}

// описание прототипов функции стрелки(линии) соединения
function Arrow(startBlock, startConnector) {
this.startBlock = startBlock
this.startConnector = startConnector
this.endBlock = null
this.endConnector = null
this.isActivated = false
this.verticalFirst = true
this.InitStart()
}
```

```
Arrow.prototype.InitStart = function () {
this.nodes = []
this.counts = []
this.AddNode(this.startBlock.x + this.startConnector.x,
this.startBlock.y + this.startConnector.y, false)
this.AppendNode(Math.sign(this.startConnector.x) * GRID_SIZE * 2,
Math.sign(this.startConnector.y) * GRID_SIZE * 2)
}
Arrow.prototype.Normalize = function (value) { return
Math.floor((value + GRID_SIZE / 2) / GRID_SIZE) * GRID_SIZE }
Arrow.prototype.AddDiagonalNode = function (x, y) {
let last = this.nodes[this.nodes.length - 1]
if (this.nodes.length > 1) {
let dx1 = x - this.nodes[this.nodes.length - 1].x
let dx2 = last.x - this.nodes[this.nodes.length - 2].x
if (Math.sign(dx1) == Math.sign(dx2)) {
this.nodes.push({ x: x, y: last.y })
return
}
} if (this.verticalFirst) { this.nodes.push({ x: last.x, y: y })
} else { this.nodes.push({ x: x, y: last.y }) }
}
Arrow.prototype.AddNode = function (x, y, normalize = true) {
if (normalize) {
x = this.Normalize(x)
y = this.Normalize(y)
} if (this.nodes.length == 0) {
this.nodes.push({ x: x, y: y })
this.counts.push(1)
return
} let dx = x - this.nodes[this.nodes.length - 1].x
let dy = y - this.nodes[this.nodes.length - 1].y
if (dx != 0 && dy != 0) {
this.counts.push(2)
this.AddDiagonalNode(x, y)
} else { this.counts.push(1) } this.nodes.push({ x: x, y: y })
}
Arrow.prototype.AppendNode = function (dx, dy, normalize = true)
{
let last = this.nodes[this.nodes.length - 1]
this.AddNode(last.x + dx, last.y + dy, normalize)
}
Arrow.prototype.RemoveNode = function () {
if (this.nodes.length == 0) return
let count = this.counts.pop()
for (let i = 0; i < count; i++)this.nodes.pop()
}
Arrow.prototype.IsIntersectBlock = function (block) {
for (let i = 1; i < this.nodes.length; i++)if
(block.IsIntersectLine(this.nodes[i - 1], this.nodes[i])) return
true
```

```
return false
}
Arrow.prototype.AddPoint = function (x, y) {
x = this.Normalize(x)
y = this.Normalize(y)
for (let i = 1; i < this.nodes.length; i++) {
let segment = this.GetSegmentInfo(i)
if (segment.x1 > x || x > segment.x2 || segment.y1 > y || y >
segment.y2) continue
if (!ADD_NODE_AS_POINT) {
let len1 = Math.abs(x - segment.x1) + Math.abs(y - segment.y1)
let len2 = Math.abs(x - segment.x2) + Math.abs(y - segment.y2)
x = 2 * x - (len1 < len2 ? segment.x1 : segment.x2)
y = 2 * y - (len1 < len2 ? segment.y1 : segment.y2)
} this.nodes.splice(i, 0, { x: x, y: y })
return
}
}
Arrow.prototype.IsConnectToBlock = function (block) {
for (let i = 0; i < block.connectors.length; i++) {
if (this.startConnector == block.connectors[i]) return true
if (this.endConnector == block.connectors[i]) return true
} return false
}
Arrow.prototype.Draw = function (ctx, x0, y0, scale, color = null)
{
if (this.nodes.length < 2) return
if (color == null) {
ctx.strokeStyle = this.isActivated ?
ARROW_ACTIVE_COLOR[DARK_THEME] : ARROW_COLOR[DARK_THEME]
ctx.fillStyle = this.isActivated ? ARROW_ACTIVE_COLOR[DARK_THEME]
: ARROW_COLOR[DARK_THEME]
} else {
ctx.strokeStyle = color
ctx.fillStyle = color
} ctx.lineWidth = ARROW_WIDTH
if (scale > 1) ctx.lineWidth *= scale
if (this.endBlock != null && this.endBlock.IsText()) {
ctx.setLineDash([GRID_SIZE / 2 * scale, GRID_SIZE / 2 * scale]) }
ctx.beginPath()
ctx.moveTo(this.nodes[0].x * scale + x0, this.nodes[0].y * scale
+ y0)
for (let i = 1; i < this.nodes.length;
i++)ctx.lineTo(this.nodes[i].x * scale + x0, this.nodes[i].y *
scale + y0)
ctx.stroke()
if (this.endBlock == null || !this.endBlock.IsText())
this.DrawEnd(ctx, x0, y0, scale)
if (this.isActivated) this.DrawNodes(ctx, x0, y0, scale)
ctx.setLineDash([])
```

```
if (this.endBlock != null && this.endBlock.IsText()) {
this.endBlock.DrawCommentLine(ctx, x0, y0, scale,
this.endConnector) }
}
Arrow.prototype.ToJSON = function (blocks) {
let startIndex = blocks.indexOf(this.startBlock)
let endIndex = blocks.indexOf(this.endBlock)
let startConnectorIndex =
this.startBlock.connectors.indexOf(this.startConnector)
let endConnectorIndex =
this.endBlock.connectors.indexOf(this.endConnector)
return { startIndex: startIndex, endIndex: endIndex,
startConnectorIndex: startConnectorIndex, endConnectorIndex:
endConnectorIndex, nodes: this.nodes, counts: this.counts }
}

// описание прототипов функции захвата элементов
function TouchEvents(diagram) {
this.diagram = diagram
let events = this
document.ontouchstart = function (e) { events.TouchStart(e) }
document.ontouchmove = function (e) { events.TouchMove(e) }
document.ontouchend = function (e) { events.TouchEnd(e) }
} TouchEvents.prototype.TouchStart = function (e) {
e.preventDefault()
this.prevTouchPoints = []
if (e.targetTouches.length == 2) {
this.prevTouchPoints.push({ x: e.targetTouches[0].clientX, y:
e.targetTouches[0].clientY - CANVAS_OFFSET })
this.prevTouchPoints.push({ x: e.targetTouches[1].clientX, y:
e.targetTouches[1].clientY - CANVAS_OFFSET })
} else if (e.targetTouches.length == 1) {
e.offsetX = e.targetTouches[0].clientX
e.offsetY = e.targetTouches[0].clientY - CANVAS_OFFSET
e.button = 0
this.diagram.MouseDown(e)
}
}
TouchEvents.prototype.GetDistance = function (p1, p2) {
let dx = p2.x - p1.x
let dy = p2.y - p1.y
return Math.sqrt(dx * dx + dy * dy)
}
TouchEvents.prototype.TouchMove = function (e) {
e.preventDefault()
if (e.targetTouches.length > 2 || e.targetTouches.length == 0)
return
if (e.targetTouches.length == 1) {
e.offsetX = e.targetTouches[0].clientX
e.offsetY = e.targetTouches[0].clientY - CANVAS_OFFSET
e.button = 0
this.diagram.MouseMove(e)
```

```
return
} let p1 = { x: e.targetTouches[0].clientX, y:
e.targetTouches[0].clientY - CANVAS_OFFSET }
let p2 = { x: e.targetTouches[1].clientX, y:
e.targetTouches[1].clientY - CANVAS_OFFSET }
let prevDistance = this.GetDistance(this.prevTouchPoints[0],
this.prevTouchPoints[1])
let currDistance = this.GetDistance(p1, p2)
if (Math.abs(currDistance - prevDistance) < 25) {
let dirX1 = Math.sign(p1.x - this.prevTouchPoints[0].x)
let dirX2 = Math.sign(p2.x - this.prevTouchPoints[1].x)
let dirY1 = Math.sign(p1.y - this.prevTouchPoints[0].y)
let dirY2 = Math.sign(p2.y - this.prevTouchPoints[1].y)
if (dirX1 == dirX2 && dirX1 != 0) {
this.diagram.x0 += Math.max(p1.x - this.prevTouchPoints[0].x, p2.x
- this.prevTouchPoints[1].x)
this.prevTouchPoints[0].x = p1.x
this.prevTouchPoints[1].x = p2.x
} if (dirY1 == dirY2 && dirY1 != 0) {
this.diagram.y0 += Math.max(p1.y - this.prevTouchPoints[0].y, p2.y
- this.prevTouchPoints[1].y)
this.prevTouchPoints[0].y = p1.y
this.prevTouchPoints[1].y = p2.y
}
} else {
this.diagram.currPoint.x = (p1.x + p2.x) / 2
this.diagram.currPoint.y = (p1.y + p2.y) / 2
if (currDistance > prevDistance) { this.diagram.UpdateScale(1) }
else { this.diagram.UpdateScale(-1) } this.prevTouchPoints[0] = p1
this.prevTouchPoints[1] = p2
}
}
TouchEvents.prototype.TouchEnd = function (e) {
e.preventDefault()
if (this.prevTouchPoints.length == 0) this.diagram.MouseUp(e)
this.prevTouchPoints = []
}

// описание прототипов функции представления рабочей области в
целом
function Diagram(canvas, input, sourceInput, textInput) {
this.canvas = canvas
this.ctx = canvas.getContext('2d')
this.WindowResize()
this.input = input
this.sourceInput = sourceInput
this.textInput = textInput
this.name = "diagram"
this.x0 = 0
this.y0 = 0
this.scales = [0.125, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 2, 3, 4, 8,
16]
```

```
this.scaleIndex = 4
this.Init()
this.InitIcons()
this.InitEvents()
this.InitMenuBlocks()
this.needTips = true
let diagram = this
setTimeout(function () { diagram.needTips = false }, TIPS_TIMEOUT)
}
Diagram.prototype.SavePicture = function () {
this.EscapeKeyDownProcess()
if (this.blocks.length == 0) {
alert("Блок-схема пуста. Сохранять нечего")
return
} let link = document.createElement("a")
link.href = this.SaveArea(this.blocks)
link.download = this.name + '.png'
link.click()
}
let canvas = document.getElementById("canvas")
let input = document.getElementById('input'); let sourceInput =
document.getElementById('source-input');    let    textInput    =
document.getElementById('text-input')
let diagram = new Diagram(canvas, input, sourceInput, textInput)
function Draw() {
diagram.Draw()
window.requestAnimationFrame(Draw)
} window.requestAnimationFrame(Draw)
</script>
```