Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных систем и сетей
Кафедра Программного обеспечения информационных технологий

К защите допустить:

Заведующая кафедрой ПОИТ

_____ Н. В. Лапицкая

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему:

ПРОГРАММНОЕ СРЕДСТВО СОЗДАНИЯ ВЕБ-ПРИЛОЖЕНИЙ С ПОМОЩЬЮ ГОТОВЫХ ГРАФИЧЕСКИХ КОМПОНЕНТОВ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ WEBIX И ЯЗЫКА ПРОГРАММИРОВАНИЯ JAVASCRIPT

БГУИР ДП 1-40 01 01 01 102 ПЗ

Студент	Н. А. Романчук
Руководитель	С. А. Медведев
Консультанты:	
от кафедры ПОИТ	С. А. Медведев
по экономической части	В. А. Палицын
Нормоконтролёр	И. М. Марина

Рецензент

Минск 2019

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

Апплет – это несамостоятельный компонент программного обеспечения, работающий в контексте другого, полновесного приложения [1].

Асинхронность (Асинхронизм) – не совпадение с чем-либо во времени; неодномоментность, неодновременность, несинхронность — характеризует процессы, не совпадающие во времени.

Грид – это двумерная система компоновки основанная на сетке, цель которой заключается в том чтобы полностью изменить способ проектирования пользовательских интерфейсов основанных на сетке [2].

Колбэк – функция обратного вызова в программировании — передача исполняемого кода в качестве одного из параметров другого кода. Обратный вызов позволяет в функции исполнять код, который задаётся в аргументах при её вызове. Этот код может быть определён в других контекстах программного кода и быть недоступным для прямого вызова из этой функции. Некоторые алгоритмические задачи в качестве своих входных данных имеют не только числа или объекты, но и действия (алгоритмы), которые естественным образом задаются как обратные вызовы (сокращение от англ. Callback: all — вызов, англ. back — обратный) [3].

Лэйаут – структурированное отображение информации на плоскость [4].

Сериализация – процесс перевода какой-либо структуры данных в последовательность битов. Обратной к операции сериализации является операция десериализации (структуризации) — восстановление начального состояния структуры данных из битовой последовательности [5].

AJAX (сокращение от англ. Asynchronous JavaScript and XML - асинхронный JavaScript и XML) – подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате, при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее [6].

Babel – это бесплатный компилятор JavaScript с открытым исходным кодом и настраиваемый транспилер, используемый в веб-разработке. Babel позволяет разработчикам программного обеспечения писать исходный код на предпочтительном языке программирования или языке разметки и переводить его Babel на JavaScript, язык, понятный современным веб-браузерам [7].

CMS (сокращение от англ. Content Management System - «система управления контентом сайта» или просто «система управления сайтом»)

– программное обеспечение, которое позволяет разрабатывать и поддерживать динамические информационные web-сайты [8].

DOM (от англ. Document Object Model — «объектная модель документа») – это не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML-, XHTML- и XML-документов, а также изменять содержимое, структуру и оформление таких документов [9].

Drag-and-drop/Drag-n-drop (в переводе с английского – тащи-и-бросай; Бери-и-Брось) – способ оперирования элементами интерфейса в интерфейсах пользователя (как графическим, так и текстовым, где элементы GUI реализованы при помощи псевдографики) при помощи манипулятора «мышь» или сенсорного экрана [10].

Git – распределённая система контроля версий, которая даёт возможность разработчикам отслеживать изменения в файлах и работать совместно с другими разработчиками. Она была разработана в 2005 году Линусом Торвальдсом, создателем Linux, для того, чтобы другие разработчики могли вносить свой вклад в ядро Linux. Git известен своей скоростью, простым дизайном, поддержкой нелинейной разработки, полной децентрализацией и возможностью эффективно работать с большими проектами [11].

UI – интерфейс, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы (сокращение от англ. User Interface – «пользовательский интерфейс») [12].

REST – архитектурный стиль взаимодействия компонентов распределенного приложения (сокращение от англ. Representational State Transfer – «передача состояния представления») [13].

SPA (сокращение от англ. Single Page Application – одностраничное приложение) – это веб-приложение или веб-сайт, использующий единственный HTML-документ как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемые HTML, CSS, JavaScript, обычно посредством AJAX [14].

Trial (триал) – Условно бесплатное программное обеспечение (программное обеспечение с безвозмездным (или возмездным при определённых условиях) использованием) [15].

UX (сокращение от англ. User Experience – «опыт пользователя») – это то, какой опыт/впечатление получает пользователь от взаимодействия с вашим интерфейсом. Удается ли ему достичь цели и насколько просто или сложно это сделать [16].

ВВЕДЕНИЕ

С появлением Web-технологиий компьютер начинают использовать совершенно новые слои населения Земли. Можно выделить две наиболее характерные группы, находящиеся на разных социальных полюсах, которые были стремительно вовлечены в новую технологию, возможно, даже помимо их собственного желания. С одной стороны, это были представители элитарных групп общества: руководители крупных организаций, президенты банков, топ- менеджеры, влиятельные государственные чиновники и т.д. С другой стороны, это были представители широчайших слоев населения: домохозяйки, пенсионеры, дети.

При появлении технологии Web компьютеры повернулись лицом к этим двум совершенно противоположным категориям потенциальных пользователей. Элиту объединяла одна черта – в силу высочайшей ответственности и практически стопроцентной занятости «большие люди» никогда не пользовались компьютером; типичной была ситуация, когда с компьютером работал секретарь. В какой-то момент времени они поняли, что компьютер им может быть полезен, что они могут результативно использовать то небольшое время, которое можно выделить на работу за компьютером. Они вдруг поняли, что компьютер – это не просто модная и дорогая игрушка, но инструмент получения актуальной информации для бизнеса. При этом им не нужно было тратить сколько-нибудь заметного времени, чтобы освоить технологию работы с компьютером (по сравнению с тем, как это было раньше).

Спектр социальных групп, подключающихся к сети Интернет и ищущих информацию в WWW, все время расширяется за счет пользователей, не относящихся к категории специалистов в области информационных технологий. Это врачи, строители, историки, юристы, финансисты, спортсмены, путешественники, священнослужители, артисты, писатели, художники. Список можно продолжать бесконечно. Любой, кто ощутил полезность и незаменимость Сети для своей профессиональной деятельности или увлечений, присоединяется к огромной армии потребителей информации во «Всемирной Паутине».

Web-технология полностью перевернула наши представления о работе с информацией, да и с компьютером вообще. Оказалось, что традиционные параметры развития вычислительной техники – производительность, пропускная способность,емкость запоминающих устройств не учитывали главного «узкого места» системы – интерфейса взаимодействия с человеком. Устаревший механизм взаимодействия человека с информационной си-

стемой сдерживал внедрение новых технологий и уменьшал выгоду от их применения. И только когда интерфейс между человеком и компьютером был упрощен до естественности восприятия обычным человеком, последовал беспрецедентный взрыв интереса к возможностям вычислительной техники.

С развитием технологий гипертекстовой разметки в Интернете стало появляться всё больше сайтов, тематика которых была совершенно различной – от сайтов крупных компаний, повествующих об успехах компании и её провалах, до сайтов маленьких фирм, предлагающих посетить их офисы в пределах одного города.

Развитие Интернет-технологий послужило толчком к появлению новой ветки в Интернете – Интернет-форумов. Стали появляться сайты, и даже целые порталы, на которых люди со всех уголков планеты могут общаться, получать ответы на любые вопросы и, даже, заключать деловые сделки.

Основные идеи современной информационной технологии базируются на концепции, согласно которой данные должны быть организованы в базы данных, с целью адекватного отображения изменяющегося реального мира и удовлетворения информационных потребностей пользователей.

Любая информационная система представляет собой программный комплекс, функции которого состоят в поддержке надежного хранения информации в памяти компьютера, выполнении специфических для данного приложения преобразований информации и/или вычислений, предоставлении пользователям удобного и легко осваиваемого интерфейса. С развитием и распространением сети Интернет информационные системы стали более интерактивными, масштабируемыми и доступными обычным пользователям.

Необходимость систем управления для владельцев сайтов начала проявляться в тот момент, когда количество материалов на веб-сайтах начало стремительно расти. Это привело к тому, что традиционные «ручные» технологии разработки и поддержки сайтов, когда сайт состоял из статических страниц и набора дополнительных специализированных скриптов, стали не успевать за быстро меняющимися условиями бизнеса. Ввод данных на сайт требовал (как минимум) знания технологий HTML/CSS верстки, изменения структуры сайтов были сопряжены с каскадным изменением большого количества взаимосвязанных страниц. Различные автоматизированные механизмы, вроде гостевых книг и новостных лент, внедренные на сайтах как отдельные скрипты и, как правило, написанные разными специалистами, перестали удовлетворять требованиям безопасности. На многих сайтах стали появляться коктейли из разных технологий и подходов к разработке,

поэтому возникла потребность в стандартизации программных решений, в разделении дизайна и содержимого на две независимые составляющие. CMS действительно разделяют сайты на две составляющие: дизайн (внешний вид сайта в целом, отдельных страниц, конкретных блоков информации) и контент. Дизайн сайта как правило «зашит» в шаблоны и изменяется значительно реже, чем контент.

Система управления сайтами – это программный комплекс, позволяющий автоматизировать процесс управления как сайтом в целом, так и сущностями в рамках сайта: макетами страниц, шаблонами вывода данных, структурой, информационным наполнением, пользователями и правами доступа, а также по возможности предоставляющий дополнительные сервисы: списки рассылки, ведение статистики, поиск, средства взаимодействия с пользователями и т. д.

Обычно системы обновления делятся на две части: внешнюю – набор HTML-страниц, генерируемых при вызове страниц из браузера посетителя сайта и внутреннюю – систему администрирования. Обе части обычно используют общее хранилище данных, в роли которого, как правило, выступает реляционная база данных (иногда встречаются другие виды хранилищ, например, XML-документы или даже текстовые файлы).

В данном дипломном проекте реализуется программное средство создания веб-приложений с помощью готовых графических компонентов, которое способно интегрироваться в другие программные решения, может использоваться как самими разработчиками для создания пользовательского интерфейса веб-приложения, так и пользователями этого веб-приложения для конфигурации пользовательского интерфейса согласно собственным предпочтениям.

Темой разрабатываемого проекта является «Программное средство создания веб-приложений с помощью готовых графических компонентов».

Перечень задач, поставленных для решения в течение преддипломной практики представлен ниже:

– Анализ предмета, постановка задачи;
– Методы и модели, положенные в основу;
– Разработка структуры проекта.

Целью дипломного проекта является разработка и реализация интегрируемого приложения, общающегося с сервером в сети Интернет с целью предоставления пользователям перечня доступных веб-компонентов. В первую очередь разрабатываемое программное средство предназначается для небольших проектов, в которые будет легко интегрироваться и при этом предоставлять возможности технологии, на которой написан.

1 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И СУЩЕСТВУЮЩИХ АНАЛОГОВ

1.1 Аналитический обзор литературных источников

Выбор литературных источников при подготовке к проектированию был обусловлен использованием определенных технологий, созданных для упрощения разработки веб-ориентированного программного обеспечения. В то же время, помимо печатных изданий, были использованы дополнительные ресурсы в сети Интернет.

Первым литературным источником, который использовался для получения информации по теме, стала книга автора Кайла Симпсона «You don’t know JS» [17]. Эта книга подробно описывает возможности и внутреннее устройство языка программирования JavaScript, а также тонкости использования предоставляемых им инструментов и работы с ним в целом.

Вторым литературным источником, который использовался для получения информации по теме, стала книга автора Кайла Симпсона «Functional-Light JavaScript» [18]. Данный источник использовался с целью получения информации о функциональном подходе к программированию для применения их в разработке программного средства. Книга является вполне практико-ориентируемой - автор простым языком объясняет основные принципы функциональной парадигмы, не вдаваясь глубоко в теорию по этой теме, что позволяет быстро разобраться в материале и начать применять полученные знания на практике в разработке программного средства.

Третьим источником информации, который использовался для получения информации по теме, является документация технологии Webix [19] - программного продукта, использовавшегося при разработке. Данная технология предоставляет большое количество уже спроектированных веб-компонентов со своей логикой работы и богатым инструментарием, что благоприятно сказывается на скорости разработки продуктов с использованием данной технологии.

Также стоит отметить сайт habr.ru, который использовался для получения информации о REST - архитектурном стиле взаимодействия компонентов распределенного приложения [13].

1.2 Анализ существующих аналогов

Целью данной курсовой работы является проектирование и разработка веб-сервиса для компоновки пользовательского интерфейса. Данное программное средство является интегрируемым сервисом, которое может ис-

пользоваться как самими разработчиками для создания пользовательского интерфейса веб-приложения, так и пользователями этого веб-приложения для конфигурации пользовательского интерфейса согласно собственным предпочтениям. Программные средства такого рода называются конструкторами сайтов или CMS [8] и обладают следующим функционалом:

– предоставление полностью готовых шаблонов сайтов с готовым дизайном;

– предоставление готовых дизайнерских решений для пользовательского интерфейса;

– гибкая система управления сайтом.

Такие сервисы целесообразно использовать для:

– создания одностраничных приложений или SPA под трафик с контекстной и таргетированной рекламой;

– визуализации идеи, чтобы впоследствии предоставить ее разработчикам и/или дизайнерам;

– быстрого запуска несложных проектов;

– тестирования идеи, чтобы понять, стоит ли тратить время и деньги на разработку;

– быстрого запуска несложных проектов;

– некоммерческих сайтов «для души».

Серьезные веб-проекты лучше создавать на зарекомендовавших себя CMS или «самописных» движках, заточенных под конкретные задачи. Это так, но в некоторых ситуациях такой подход слишком долог, дорог и трудозатратен. С тем же WordPress нужно разбираться несколько недель. Если есть время и желание изучать тонкости самостоятельно или средства для оплаты услуг специалиста - отлично. В противном случае можно воспользоваться визуальными конструкторами. Это не панацея, есть проекты, которые невозможно реализовать без участия дизайнеров и программистов.

Выбирать конструктор стоит исходя из конкретных задач. Некоторые отлично справляются с Landing Page, другие - подходят для создания многостраничных сайтов, третьи хорошо продвигаются в поиске. Давайте сравним популярные сервисы, чтобы понять в какой ситуации лучше использовать тот или иной продукт.

При выборе инструмента для создания сайта нужно учитывать много параметров. Они зависят от типа ресурса и задач, которые он должен решать. Для удобства я составила сравнительную таблицу. В ней прописаны важные, на мой взгляд, характеристики и функциональные особенности конструкторов, которые упомянуты в этом обзоре. Сравним сервисы по таким параметрам:

– типы сайтов - на какие ресурсы рассчитан функционал (визитка, лендинг, магазин и т.д.);

– уровень пользователей - опыт в разработке: новички, продвинутые пользователи, профессионалы;

– адаптивность шаблонов - наличие в каталоге макетов, адаптированных для мобильных устройств;

– количество готовых шаблонов - сколько в каталоге готовых макетов, за которые не придется платить отдельно;

– уровень пользователей - опыт в разработке: новички, продвинутые пользователи, профессионалы;

– адаптивность шаблонов - наличие в каталоге макетов, адаптированных для мобильных устройств;

– количество готовых шаблонов - сколько в каталоге готовых макетов, за которые не придется платить отдельно;

– уровень кастомизации шаблонов - возможность изменения элементов и дизайна в целом: высокая, средняя, низкая;

– возможность создать сайт с нуля - можно ли открыть пустой макет и собрать страницы из блоков и виджетов;

– обучающие материалы - информационная база по пользованию конструктором;

– возможность редактировать и добавлять код - добавлять свои элементы и редактировать стили через HTML и CSS;

– бесплатный тариф - наличие бесплатного тарифа. Ограничения прописаны в детальном обзоре каждого конструктора ниже;

– триал - наличие тестового периода с расширенным функционалом.

– техподдержка - язык и способы поддержки пользователей;

– минимальный тариф - стоимость минимального тарифа;

– способы оплаты - варианты оплаты тарифов и дополнительных услуг;

– интеграции - подключение к сторонним сервисам для расширения функционала сайта;

– домен - возможность подключить свой домен на бесплатном тарифе, а также условия, на которых он предоставляется бесплатно;

– SEO - возможности оптимизации сайта;

– импорт/экспорт товаров - способы загрузки большого количества товаров в каталог;

– интеграция с CRM - подключение к CRM для автоматического импорта заказов с сайта;

– интеграция с системами аналитики - возможность подключиться к

сторонним сервисам аналитики;

– онлайн-оплата - платежные системы, которые можно подключить к сайту;

– интеграция с соцсетями - кнопки, виджеты и комментарии.

В качестве рассматриваемых аналогов будут выступать следующие решения:

– Tilda Publishing;

– LPgeneratorWIX.

1.2.1 Tilda Publishing

Tilda – интуитивный конструктор сайтов. Подходит для создания небольших проектов - информационных и корпоративных ресурсов, Landing Page и интернет-магазинов с десятком-другим позиций. Хотя для последних есть более удобные решения, за последний год Tilda добавила множество возможностей для этого типа сайтов. Появились полноценная корзина с вариантами доставки и оплаты, блоки карточек товаров со встроенными попапами, в которых отображаются увеличенные фотографии и расширенное описание, лейблы «хит», акция и другие для визуального выделения товаров. Кроме того, сайт интегрируется с несколькими платежными системами, интернет-эквайерами и CRM. Заявки можно отслеживать во встроенном инструменте конструктора, экспортировать в Google Sheets или Telegram.

Функционал Tilda ориентирован на эффектное оформление лонгридов - стильная типографика, много блоков для комбинирования текстового, визуального и видео-контента. Возможности не ограничиваются готовыми шаблонами и блоками - и то, и другое можно разработать самостоятельно с нуля, используя конструкторы. Для тех, у кого на это нет времени и желания, в каталоге около 200 дизайнов, которые можно настроить под себя с помощью 450 блоков. Есть готовые макеты для пиццерий, антикафе, салонов красоты и других ниш. Для визуализации отдельных элементов сайта дизайнеры Тильды отрисовали целую библиотеку иконок под разные сферы бизнеса, которая регулярно пополняется [20].

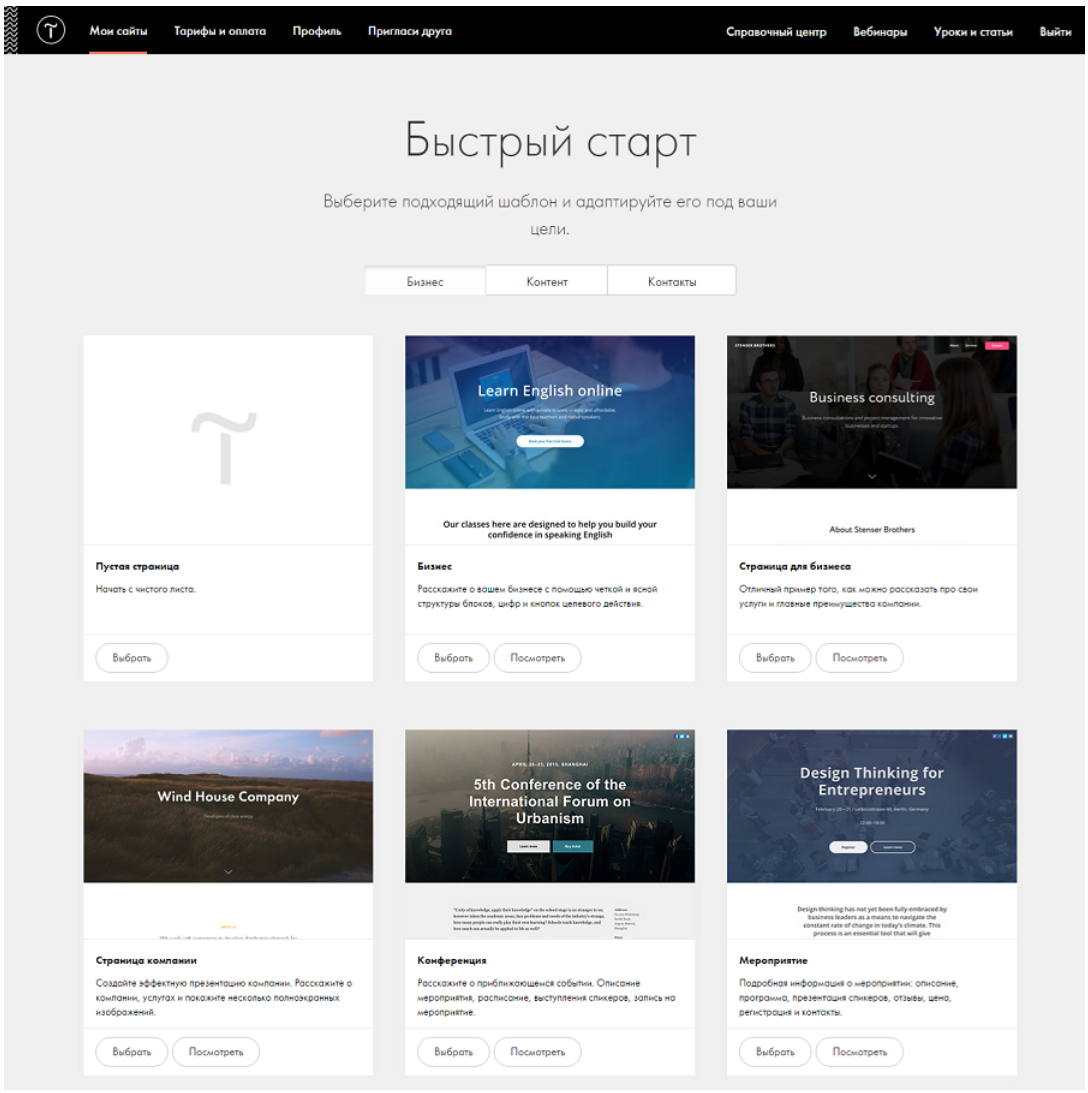

Рисунок 1.1 – Скриншот приложения «Tilda Publishing»

Достоинства:

– Большой выбор готовых шаблонов;

– Удобный интуитивно понятный интерфейс;

– Широкие возможности для кастомизации - цвета, шрифты, отступы, прозрачность, анимация;

– Стильные адаптивные шаблоны;

– Возможность отдельно настроить отступы для мобильных и десктопных версий;

– Возможность создать свой уникальный макет с нуля;

– Большой выбор модулей - текст, блоки преимуществ, формы заявок и обратной связи, опросов, онлайн-бронирования и т. д;

– Генератор UTM-меток;

– Обучающие статьи и видео-уроки по работе с сервисом;

– Хорошая типографика, можно подключать шрифты из Google Fonts,

Typekit и собственной коллекции;

– Возможность добавить свои элементы с помощью HTML, CSS и JS;

– Интеграция с CRM, сервисами обратных звонков, онлайн-чатами и электронными платежными системами;

– Сайт можно перенести на другой хостинг или собственный сервер, а также интегрировать созданные на Tilda страницы с проектами на Битрикс, WordPress и другими;

– Встроенная аналитика, подключение Google Analytics и «Яндекс.Метрики»;

– Отслеживание количества кликов по кнопкам, заполнения форм и других событий в «Метрике» и «Аналитике» по умолчанию, без настройки целей;

– Подключение на сайт ленты постов Instagram с автоматическим обновлением;

– Возможность настроить передачу данных в налоговую в соответствии с 54-ФЗ;

– Конструктор политики обработки персональных данных.

– Возможность сохранить и использовать для дальнейшей работы собственные шаблоны;

– Подключение HTTPS в интерфейсе конструктора.

– Функционал мультилендинга - можно создать динамический контент, чтобы внешний вид сайта и информация подстраивались под конкретного пользователя;

– Интеграция с популярными онлайн-банками - «Сбербанк», «Альфа Банк» и «Тинькофф», чтобы принимать платежи и получать деньги на расчетный счет;

– Функционал для верстки email-рассылок с последующей отправкой через MailChimp, UniSender или любой другой сервис с помощью экспорта кода.

Недостатки:

– Высокие тарифы;

– Для хорошего результата требуются определенные познания в вебдизайне.

1.2.2 LPgenerator

LPgenerator – мощный сервис для разработки одностраничных сайтов. Узкая специализация с лихвой окупается множеством возможностей и интеграций. Не подойдет для создания визитки, блога, информационного портала или онлайн-гипермаркета. Есть возможность подключить на од-

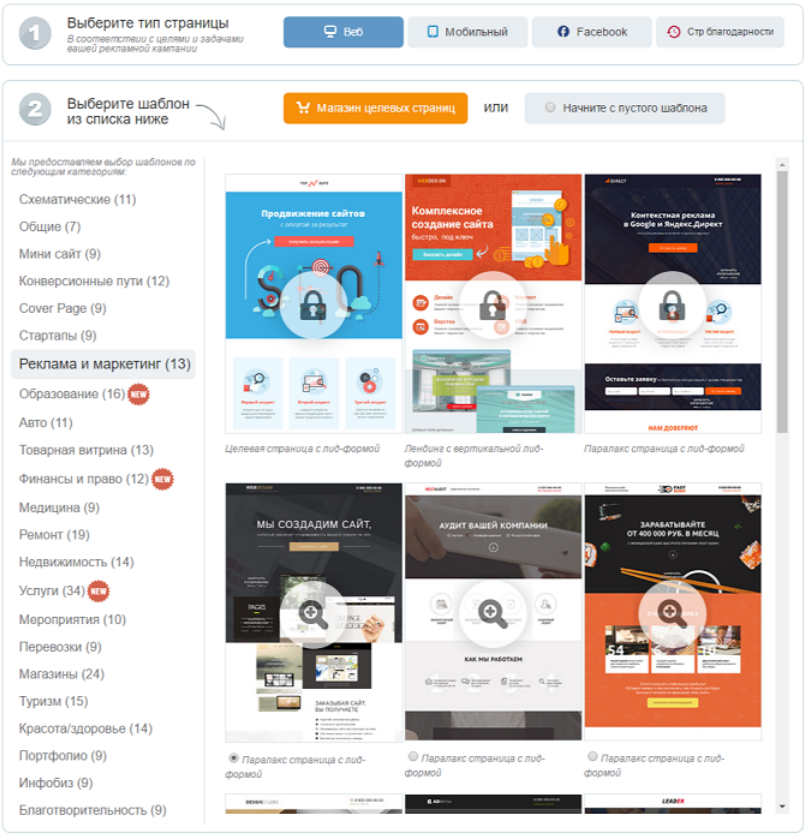

Рисунок 1.2 – Каталог шаблонов «LPgenerator»

ностраничник витрину, но этот инструмент скорее дополнительный. Зато лендинг при должных навыках получится что надо.

Сервис ориентирован на профессионалов. Разработчики заинтересованы в том, чтобы на их платформе создавались красивые и конверсионные сайты. Об этом говорит все - от стоимости и содержания тарифов до уровня тех. поддержки и обучения новых пользователей.

В LPG есть все для создания уникальных профессиональных сайтов - множество красивых современных шаблонов вкупе с широкими возможностями кастомизации, сборка макета из блоков с чистого листа и доступ к HTML/CSS коду. Но главное отличие от других конструкторов заключается в дополнительных инструментах для раскрутки и заработка на лендингах - CRM и работа с лидами, подробная и детальная аналитика, управление источниками трафика и многое другое [20].

2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

2.1 Разработка функциональных требований

Основной целью создаваемого программного средства является предоставление возможности создать собственные веб-компоненты на основании предоставляемых платформой или разработчиком, который интегрирует данное программное средство в свой продукт, так называемый аплет [1], а также скомпоновать из готовых «односложных» компонентов более комплексные, задав им при этом логику взаимодействия.

Пользователями данного программного средства могут быть как разработчики с целью более быстрой разработки интерфейса взаимодействия с пользователя с приложением, так и пользователи,которым разработчики, интегрирующие данное программное средство в свой продукт, пожелают предоставить возможность самим определить, как будет выглядеть для них дизайн веб-приложения. Программное средство предоставляет интерфейс для взаимодействия с ним, позволяя гибко себя сконфигурировать. Однако, существует ряд требований к разработчику, интегрирующему данное программное средство в свой продукт:

– Совместимость технологии, на которой осуществляется разработка продукта с технологией Webix (React, Angular, Vue JS);

– Разработчик должен задать место, куда будет интегрироваться программное средство;

– Разработчик должен предоставить способ получения конфигурационных объектов веб-компонентов, которые будут доступны пользователю изначально, в объект конфигурации (JSON, представляющий собой массив, содержащий объекты конфигураций компонентов или ссылку, по которой приложение сможет получить данный JSON).

Перечислим требования к программному средству:

– Стабильная работа;

– Высокая скорость вычислений;

– Простота интеграции в готовые программные решения;

– Дружелюбный интерфейс;

– Возможность кастомизации;

– Возможность протестировать макет в песочнице;

– Возможность сохранять итоговый объект конфигурации скомпонованного пользователем макета;

– Обработка исключительных ситуаций и вывод информации о любых ошибках пользователю.

В качестве системы контроля версий был выбран Git [11], а в качестве GUI интерфейса для работы с Git используется программное средство GitKraken, так как обладает удобным, доступным интерфейсом, позволяющим легко ориентироваться в возможностях данного программного средства.

2.2 Выбор языка программирования

Для разработки программного средства создания веб-приложений с помощью готовых графических компонентов был выбран язык программирования JavaScript и технология Webix - библиотека для написания UI приложения.

Гипертекстовая информационная система состоит из множества информационных узлов, множества гипертекстовых связей, определенных на этих узлах и инструментах манипулирования узлами и связями. Технология World Wide Web - это технология ведения гипертекстовых распределенных систем в Internet , и, следовательно, она должна соответствовать общему определению таких систем. Это означает, что все перечисленные выше компоненты гипертекстовой системы должны быть и в Web.

Web, как гипертекстовую систему, можно рассматривать с двух точек зрения. Во-первых, как совокупность отображаемых страниц, связанных гипертекстовыми переходами (ссылками - контейнер ANCHOR). Во-вторых, как множество элементарных информационных объектов, составляющих отображаемые страницы (текст, графика, мобильный код и т.п.). В последнем случае множество гипертекстовых переходов страницы - это такой же информационный фрагмент, как и встроенная в текст картинка.

При втором подходе гипертекстовая сеть определяется на множестве элементарных информационных объектов самими HTML-страницами, которые и играют роль гипертекстовых связей. Этот подход более продуктивен с точки зрения построения отображаемых страниц «на лету» из готовых компонентов.

При генерации страниц в Web возникает дилемма, связанная с архитектурой «клиент-сервер». Страницы можно генерировать как на стороне клиента, так и на стороне сервера. В 1995 году специалисты компании Netscape создали механизм управления страницами на клиентской стороне, разработав язык программирования JavaScript.

Таким образом, JavaScript - это язык управления сценариями просмотра гипертекстовых страниц Web на стороне клиента. Если быть более точным, то JavaScript - это не только язык программирования на стороне

клиента. Liveware, прародитель JavaScript, является средством подстановок на стороне сервера Netscape. Однако наибольшую популярность JavaScript обеспечило программирование на стороне клиента.

Основная идея JavaScript состоит в возможности изменения значений атрибутов HTML-контейнеров и свойств среды отображения в процессе просмотра HTML-страницы пользователем. При этом перезагрузки страницы не происходит.

На практике это выражается в том, что можно, например, изменить цвет фона страницы или интегрированную в документ картинку, открыть новое окно или выдать предупреждение.

Название «JavaScript» является собственностью Netscape. Реализация языка, осуществленная разработчиками Microsoft, официально называется Jscript. Версии JScript совместимы (если быть совсем точным, то не до конца) с соответствующими версиями JavaScript, т.е. JavaScript является подмножеством языка JScript.

JavaScript стандартизован ECMA (EuropeanComputerManufacturers-Association - Ассоциация европейских производителей компьютеров). Соответствующие стандарты носят названия ECMA-262 и ISO-16262. Этими стандартами определяется язык ECMAScript, который примерно эквивалентен JavaScript 1.1. Отметим, что не все реализации JavaScript на сегодня полностью соответствуют стандарту ECMA. В рамках данного курса мы во всех случаях будем использовать название JavaScript.

JavaScript - объектно-ориентированный скриптовый язык программирования. Является диалектом языка ECMAScript.

JavaScript обычно используется как встраиваемый язык для программного доступа к объектам приложений. Наиболее широкое применение находит в браузерах как язык сценариев для придания интерактивности вебстраницам.

Основные архитектурные черты: динамическая типизация, слабая типизация, автоматическое управление памятью, прототипное программирование, функции как объекты первого класса.

На JavaScript оказали влияние многие языки, при разработке была цель сделать язык похожим на Java, но при этом лёгким для использования непрограммистами. Языком JavaScript не владеет какая-либо компания или организация, что отличает его от ряда языков программирования, используемых в веб-разработке.

Название «JavaScript» является зарегистрированным товарным знаком компании Oracle Corporation [21].

2.2.1 Возможности языка программирования JavaScript

JavaScript обладает рядом свойств объектно-ориентированного языка, но реализованное в языке прототипирование обусловливает отличия в работе с объектами по сравнению с традиционными объектно-ориентированными языками. Кроме того, JavaScript имеет ряд свойств, присущих функциональным языкам - функции как объекты первого класса, объекты как списки, карринг, анонимные функции, замыкания - что придаёт языку дополнительную гибкость.

Несмотря на схожий с C синтаксис, JavaScript по сравнению с языком Си имеет коренные отличия:

– объекты, с возможностью интроспекции;

– функции как объекты первого класса;

– автоматическое приведение типов;

– автоматическая сборка мусора;

– анонимные функции.

В языке отсутствуют такие полезные вещи, как:

– модульная система: JavaScript не предоставляет возможности управлять зависимостями и изоляцией областей видимости;

– стандартная библиотека: в частности, отсутствует интерфейс программирования приложений по работе с файловой системой, управлению потоками ввода/вывода, базовых типов для бинарных данных;

– стандартные интерфейсы к веб-серверам и базам данных;

– система управления пакетами, которая бы отслеживала зависимости и автоматически устанавливала их.

Синтаксис языка JavaScript во многом напоминает синтаксис Си и Java, семантически же язык гораздо ближе к Self, Smalltalk или даже Лиспу.

В JavaScript:

– все идентификаторы регистрозависимы;

– в названиях переменных можно использовать буквы, подчёркивание, символ доллара, арабские цифры;

– названия переменных не могут начинаться с цифры;

– для оформления однострочных комментариев используются //, многострочные и

– внутристрочные комментарии начинаются с /* и заканчиваются */.

Для создания механизма управления страницами на клиентской стороне было предложено использовать объектную модель документа. Суть модели в том, что каждый HTML-контейнер — это объект, который характеризуется тройкой:

– свойства;

– методы;
– события.

Объектную модель можно представить как способ связи между страницами и браузером. Объектная модель — это представление объектов, методов, свойств и событий, которые присутствуют и происходят в программном обеспечении браузера, в виде, удобном для работы с ними кода HTML и исходного текста сценария на странице. Мы можем с ее помощью сообщать наши пожелания браузеру и далее — посетителю страницы. Браузер выполнит наши команды и соответственно изменит страницу на экране.

Объекты с одинаковым набором свойств, методов и событий объединяются в классы однотипных объектов. Классы — это описания возможных объектов. Сами объекты появляются только после загрузки документа браузером или как результат работы программы. Об этом нужно всегда помнить, чтобы не обратиться к объекту, которого нет.

2.3 Используемые стандарты

ECMAScript — это встраиваемый расширяемый не имеющий средств ввода-вывода язык программирования, используемый в качестве основы для построения других скриптовых языков. Стандартизирован международной организацией ECMA в спецификации ECMA-262. Расширения языка: JavaScript, JScript и ActionScript [22].

2.3.1 Что такое ECMAScript

Сначала немного истории. JavaScript создавался как скриптовый язык для Netscape. После чего он был отправлен в ECMA International для стандартизации (ECMA — это ассоциация, деятельность которой посвящена стандартизации информационных и коммуникационных технологий). Это привело к появлению нового языкового стандарта, известного как ECMAScript [23].

Последующие версии JavaScript уже были основаны на стандарте ECMAScript. Проще говоря, ECMAScript — стандарт, а JavaScript — самая популярная реализация этого стандарта.

2.3.2 История версий

ES — это просто сокращение для ECMAScript. Каждое издание ECMAScript получает аббревиатуру ES с последующим его номером. Всего существует 8 версий ECMAScript. ES1 была выпущена в июне 1997 года, ES2 — в июне 1998 года, ES3 — в декабре 1999 года, а версия ES4 — так и

не была принята. Не будем углубляться в эти версии, так как они морально устарели, а рассмотрим только последние четыре.

ES5 был выпущен в декабре 2009 года, спустя 10 лет после выхода третьего издания. Среди изменений можно отметить:

– поддержку строгого режима (strict mode);

– аксессоры getters и setters;

– возможность использовать зарезервированные слова в качестве ключей свойств и ставить запятые в конце массива;

– многострочные строковые литералы;

– новую функциональность в стандартной библиотеке;

– поддержку JSON.

Версия ES6/ES2015 вышла в июне 2015 года. Это также принесло некую путаницу в связи с названием пакета, ведь ES6 и ES2015 — это одно и то же. С выходом этого пакета обновлений комитет принял решение перейти к ежегодным обновлениям. Поэтому издание было переименовано в ES2015, чтобы отражать год релиза. Последующие версии также называются в соответствии с годом их выпуска. В этом обновлении были сделаны следующие изменения:

– добавлено деструктурирующее присваивание;

– добавлены стрелочные функции;

– в шаблонных строках можно объявлять строки с помощью ‘ (обратных кавычек). Шаблонные строки могут быть многострочными, также могут интерполироваться;

– let и const — альтернативы var для объявления переменных. Добавлена «временная мертвая зона»;

– итератор и протокол итерации теперь определяют способ перебора любого объекта, а не только массивов. Symbol используется для присвоения итератора к любому объекту;

– добавлены функции-генераторы. Они используют yield для создания последовательности элементов. Функции-генераторы могут использовать yield* для делегирования в другую функцию генератора, кроме этого они могут возвращать объект генератора, который реализует оба протокола;

– добавлены промисы.

ES2016 (ES7) вышла в июне 2016 года. Среди изменений в этой версии ECMAScript можно отметить:

– оператор возведения в степень **;

– метод Array.prototype.includes, который проверяет, содержится ли переданный аргумент в массиве.

Спустя еще год выходит версия ES2017 (ES8). Данный стандарт по-

лучил следующие изменения:

– асинхронность теперь официально поддерживается (async/await);

– добавлена возможность ставить запятые в конце списка аргументов функций;

– добавлено два новых метода для работы со строками: padStart() и padEnd(). Метод padStart() подставляет дополнительные символы слева, перед началом строки. А padEnd(), в свою очередь, справа, после конца строки;

– добавлена функция Object.getOwnPropertyDescriptors(), которая возвращает массив с дескрипторами всех собственных свойств объекта;

– добавлено разделение памяти и объект Atomics.

– Что же касается ES.Next, то этот термин является динамическим и автоматически ссылается на новую версию ECMAScript. Стоит отметить, что каждая новая версия приносит новые функции для языка

Выводы:

– ECMAscript выходит ежегодно;

– первые пакеты обновления назывались ES1, ES2, ES3, ES4, ES5;

– новые выпуски (начиная с 2015 года) получили название ES2015, ES2016, ES2017 (аббревиатура ES + год выпуска);

– ECMAScript является стандартом, а JavaScript — это самая популярная реализация этого стандарта.

2.4 Выбор технологии разработки пользовательского интерфейса

Webix – это JavaScript и HTML5 UI библиотека, при помощи которой веб-разработчики могут создавать качественные кросс-браузерные и кросс-платформенные веб-приложения с отзывчивым дизайном. Библиотека доступна под двумя лицензиями: GNU GPLv3 и коммерческой.

Библиотека предлагает 70 готовых к использованию ui-виджетов, которые с легкостью настраиваются в соответствии с требованиями вашего проекта. Webix предлагает простую интеграцию с JQuery, Backbone, Angular и может работать с любой серверной платформой, например, PHP, .NET, Java и Ruby [24].

Особенности:

– легкость освоения. Документация довольно подробна, и понять, как все устроено, несложно;

– интеграция с популярными фреймворками. Реализована интеграция с Backbone.js, AngularJS и jQuery. Последняя фича, например, позволяет

создавать Webix-виджеты с использованием jQuery-синтаксиса;

– интеграция со сторонними виджетами. В этом пункте ограничимся списком: Mercury, Nicedit, Tinymce, CodeMirror, CKEditor, Raphael, D3, Sigma, JustGage, Google Maps, Nokia Maps, Yandex Maps, dhtmlxScheduler and dhtmlxGantt;

– размер — маленький, скорость — высокая. В сжатом виде .js-файл весит всего 128 КБ, и при этом все работает довольно-таки быстро (по словам разработчиков так и вовсе «летает»);

– поддержка тачскрина. Созданные виджеты одинаково хорошо себя чувствуют как на десктопах, так и на смартфонах/планшетах.

2.5 Диаграмма прецедентов приложения

Диаграмма прецедентов в UML диаграмма, отражающая отношения между актёрами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Прецедент, также: вариант использования, сценарий использования — спецификация последовательностей действий (варианты последовательностей и ошибочные последовательности) в Унифицированном языке моделирования (UML), которые может осуществлять система, подсистема или класс, взаимодействуя с внешними действующими лицами.

Сформированная модель вариантов использования разрабатываемой системы показана на рисунке 2.1.

Основным актером, взаимодействующим с системой, является «Пользователь», который взаимодействует с программным средством создания веб-приложений с помощью готовых графических компонентов. Данный пользователь, используя графический интерфейс данного программного средства, получает доступ к его функциям. В качестве пользователя может выступать как конечный пользователь приложения, в которое будет интегрироваться данное программное средство, или сам разработчик в процессе разработки своего программного средства.

Ограничение функциональности для пользователей могут определять разработчики, интегрирующие данный программный модуль свое программное средство. Да и потом, сами разработчики могут в процессе разработки пользоваться данным инструментом с целью ускорения процесса разработки путем автоматизации тривиальных действий создания графический компонентов.

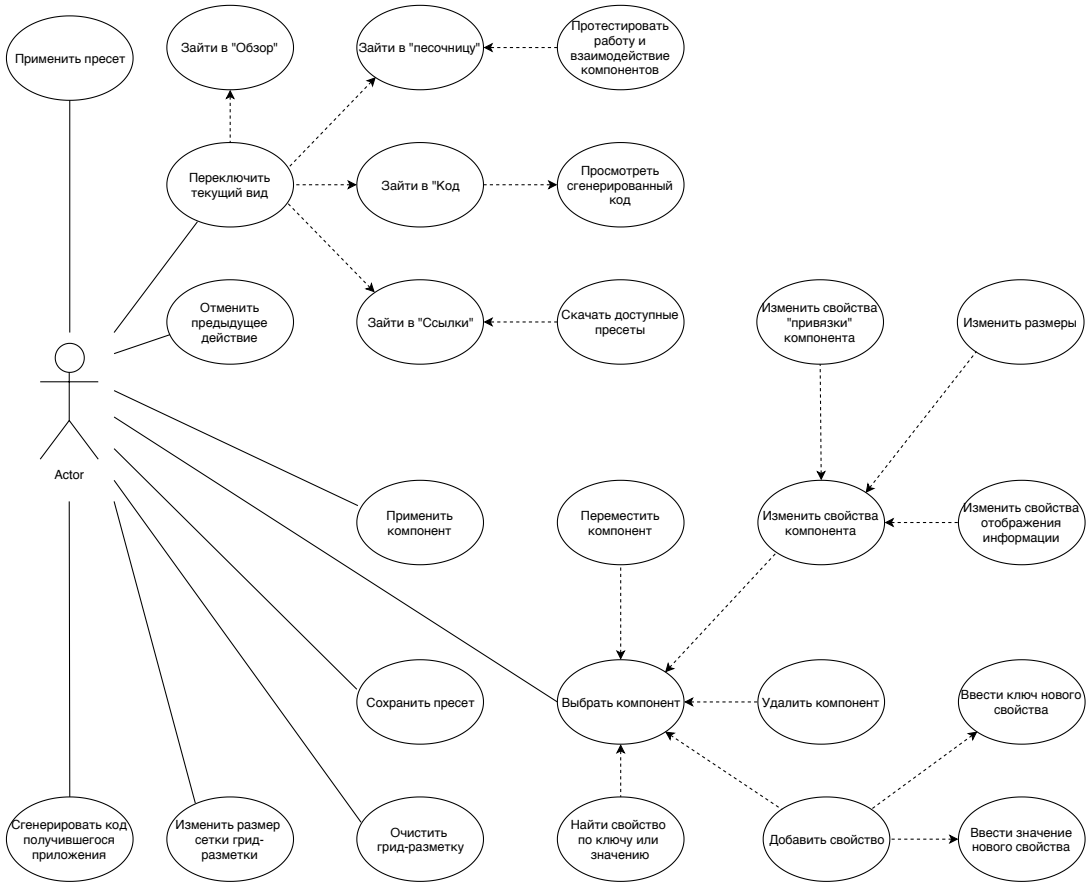

Рисунок 2.1 – Диаграмма прецедентов приложения

На основании диаграммы прецедентов программного средства, изображенной на рисунке 2.1 можно выделить следующие сценарии его использования:

– применение пресета из списка доступных;

– очистить грид-разметку;

– отменить предыдущее действие;

– переключить текущий вид, используя меню;

– изменить размер сетки грид-разметки;

– применить компонент путем перетаскивания нужного из списка компонентов на грид-разметку;

– сохранить пресет;

– сгенерировать код приложения;

– выбрать компонент и применить в его отношении изменения.

Таким образом можно сделать вывод, что программное средство представляет собой довольно гибкий и многофункциональный инструмент. Однако определять доступность всей функциональности конечному пользователю будут разработчики, интегрирующие данное программное средство в свой продукт: ведь не каждому пользователю нужно настолько глубокая

кастомизация приложения - большинству хватит просто расположить графические компоненты согласно собственным предпочтениям, не вдаваясь в подробности «связывания» компонентов.

Гибкость в определении доставляемой пользователю функциональности является большим преимуществом данного программного средства.

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Проектирование – процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или её части. Результатом проектирования является проект – целостная совокупность моделей, свойств или характеристик, описанных в форме, пригодной для последующей реализации системы. Оно, наряду с анализом требований, является частью большой стадии жизненного цикла системы, называемой определением системы.

Проектирование системы направлено на представление системы, соответствующее предусмотренной цели, принципам и замыслам; оно включает оценку и принятие решений по выбору таких компонентов системы, которые отвечают её архитектуре и укладываются в предписанные ограничения.

3.1 Описание используемых паттернов проектирования

Шаблон проектирования или паттерн в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

«Низкоуровневые» шаблоны, учитывающие специфику конкретного языка программирования, называются идиомами. Это хорошие решения проектирования, характерные для конкретного языка или программной платформы, и потому не универсальные.

На наивысшем уровне существуют архитектурные шаблоны, они охватывают собой архитектуру всей программной системы. Алгоритмы по своей сути также являются шаблонами, но не проектирования, а вычисления, так как решают вычислительные задачи. В сравнении с полностью самостоятельным проектированием, шаблоны обладают рядом преимуществ. Основная польза от использования шаблонов состоит в снижении сложности разработки за счёт готовых абстракций для решения целого класса проблем.

Шаблон даёт решению своё имя, что облегчает коммуникацию между разработчиками, позволяя ссылаться на известные шаблоны. Таким образом, за счёт шаблонов производится унификация деталей решений: модулей, элементов проекта, — снижается количество ошибок. Применение шаблонов концептуально сродни использованию готовых библиотек кода.

Правильно сформулированный шаблон проектирования позволяет, отыскав удачное решение, пользоваться им снова и снова. Набор шаблонов помогает разработчику выбрать возможный, наиболее подходящий вариант проектирования.

Хотя легкое изменение кода под известный шаблон может упростить понимание кода, по мнению Стива Макконнелла, с применением шаблонов могут быть связаны две сложности. Во-первых, слепое следование некоторому выбранному шаблону может привести к усложнению программы. Во-вторых, у разработчика может возникнуть желание попробовать некоторый шаблон в деле без особых оснований.

3.1.1 Одиночка

В данном программном средстве используется паттерн «Одиночка». Одиночка (англ. Singleton) — порождающий шаблон проектирования, гарантирующий, что в однопроцессном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру [25]. У класса есть только один экземпляр, и он предоставляет к нему глобальную точку доступа.

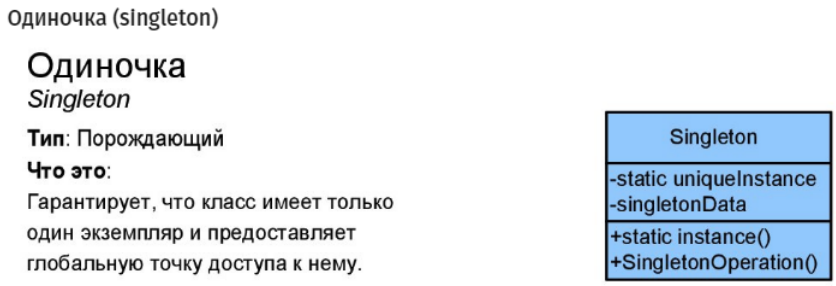

Рисунок 3.1 – Паттерн проектирования «Одиночка»

Существенно то, что можно пользоваться именно экземпляром класса, так как при этом во многих случаях становится доступной более широкая функциональность. Например, к описанным компонентам класса можно обращаться через интерфейс, если такая возможность поддерживается языком.

Глобальный «одинокий» объект — именно объект, а не набор процедур, не привязанных ни к какому объекту — бывает нужен:

– если используется существующая объектно-ориентированная библиотека;

– если есть шансы, что один объект когда-нибудь превратится в несколько;

– если интерфейс объекта (например, игрового мира) слишком сложен и не стоит засорять основное пространство имён большим количеством функций;

– если, в зависимости от каких-нибудь условий и настроек, создаётся один из нескольких объектов. Например, в зависимости от того, ведётся лог или нет, создаётся или настоящий объект, пишущий в файл, или «заглушка», ничего не делающая.

Минусы:

– если объект нужен уже при инициализации, он может быть затребован раньше, чем будет создан;

– бывает, что объект нужен не всегда. В таком случае его создание можно пропустить.

3.1.2 Фабрика

Factory - это паттерн создания объектов (creational pattern) [25]. Данный шаблон проектирования предоставляет интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс инстанциировать.

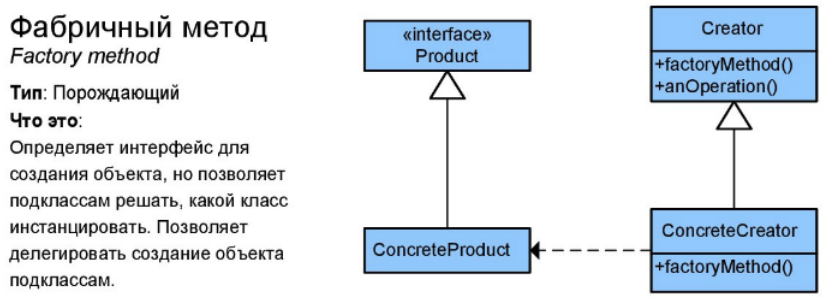

Рисунок 3.2 – Паттерн проектирования «Фабрика»

Иными словами, Фабрика делегирует создание объектов наследникам родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

3.2 Проектирование алгоритма работы программы

На рисунке 3.3 представлена схема алгоритма работы программного средства создания веб-приложений с помощью готовых графических компонентов. На ней отражены такие возможные действия, доступные для выполнения, как применение пресетов и компонентов, редактирование свойств компонентов, удаление компонентов.

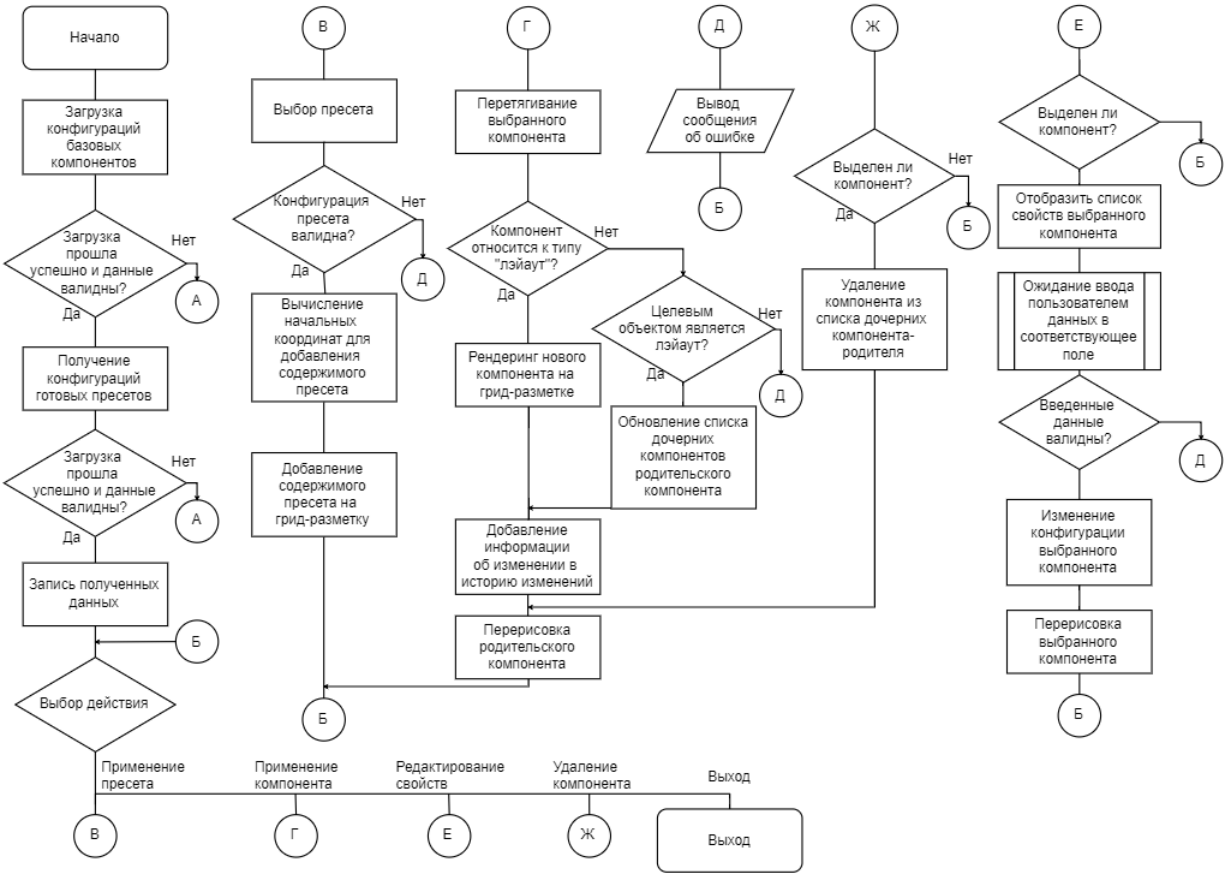

Рисунок 3.3 – Диаграмма прецедентов приложения

В зависимости от конкретного действия, каждое содержит в себе собственные шаги реализации. Так при выборе действия «Применение компонента» происходит отрисовка одного компонента, перетянутого пользователем на грид-разметку, а при выборе «Применение пресета» происходит отрисовка целой совокупности компонентов. «Редактирование свойств» содержит в себе проверку введённых данных на корректность и процесс измненения свойств выбранного компонента. «Удаление компонента» осуществляется при выбора компонента и нажатии кнопки удаления, в таком случае все дочерние элементы (если они есть) также удаляются, после чего родительский компонент (если выбранный удаляемый компонент не был помещен непосредственно на грид-разметку) также перерисовывается. Каж-

дое добавление компонента влечет за собой дополнительное действие в виде запоминания выполненного действия в истории изменений, чтобы в случае необходимости «откатить» последнее добавление элемента.

3.3 Диаграмма последовательности программного средства

Диаграмма последовательности (Sequence Diagram) – диаграмма, на которой показаны взаимодействия объектов, упорядоченные по времени их проявления.

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники), вертикальные линии, отображающие течение времени при деятельности объекта,и стрелки, показывающие выполнение действий объектами. На данной диаграмме объекты располагаются слева направо. Ее недостатком является то, что она занимает много места.

Диаграммы последовательности, описывающие сценарии Business Use Case в виде последовательности обмена сообщениями между объектами - действующими лицами и объектами-исполнителями. Такие диаграммы помогают явно определить в модели обязанности каждого исполнителя в виде набора операций класса.

На диаграмме последовательности изображаются только те объекты, которые непосредственно участвуют во взаимодействии. Ключевым моментом для диаграмм последовательности является динамика взаимодействия объектов во времени.

В UML диаграмма последовательности имеет как бы два измерения. Первое слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия. Правее изображается другой объект, который непосредственно взаимодействует с первым. Таким образом, все объекты на диаграмме последовательности образуют некоторый порядок, определяемый очередностью или степенью активности объектов при взаимодействии друг с другом.

На рисунке 3.4 представлена диаграмма последовательности применения предопределенных конфигураций графических компонентов программного средства создания веб-приложений с помощью готовых графических компонентов. На ней отражен процесс, происходящий в программе каждый раз при перетягивании пользователем компонента на грид.

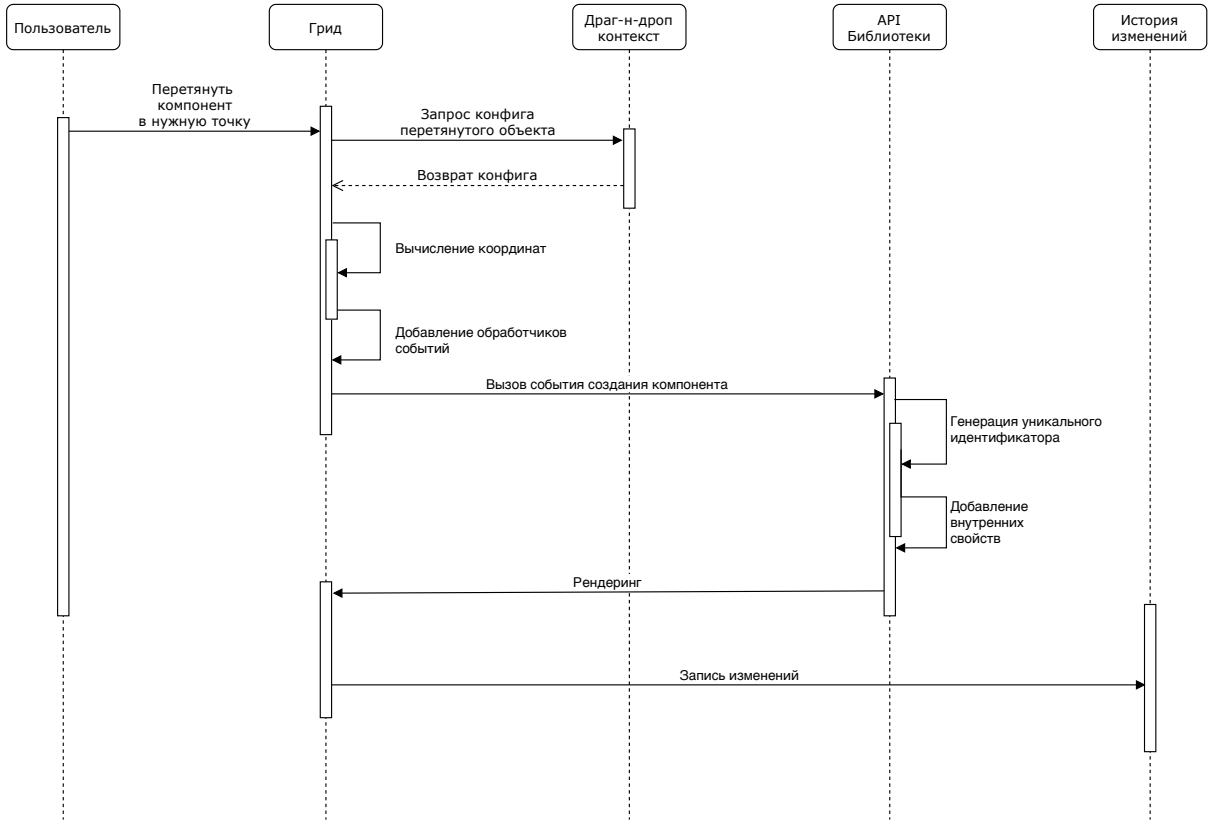

Рисунок 3.4 – Диаграмма последовательности использования предопределенных конфигураций графических компонентов

3.4 Разработка архитектуры программного средства

В данной архитектуре «Frontend» частью является клиентское приложение. Эта часть системы обменивается данными с серверной, отправляя запросы пользователя в виде HTTP запросов и получая ответы в виде структурированной структуры данных.

Программное средство создания веб-приложений с помощью готовых графических компонентов является интегрируемым сервисом на стороне клиентского приложения, а не самостоятельным программным средством. Таким образом, разработка общей архитектуры фронтенд части приложения остается за разработчиком, интегрирующим данное программное средство в свой продукт.

Данное программное средство является SPA-приложением, осуществляющим логическую навигацию между своими страницами. Общая схема работы одностраничных приложений показана на рисунке 3.5.

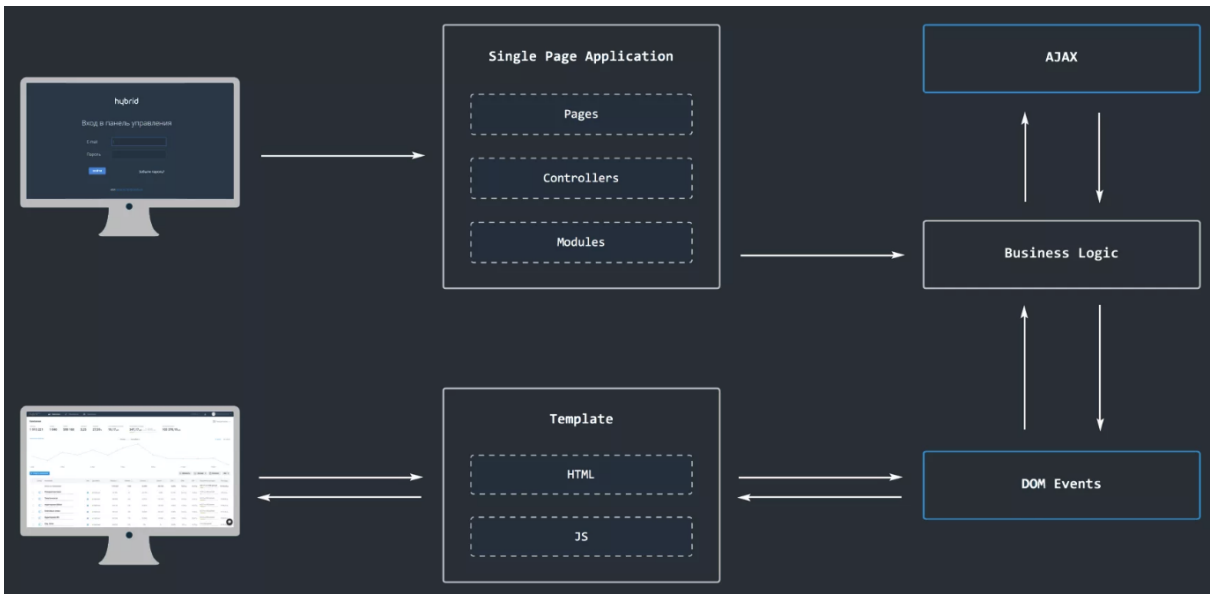

Рисунок 3.5 – Диаграмма последовательности использования предопределенных конфигураций графических компонентов

Программное средство состоит из следующих страниц:

– обзорная страница, скрывающая от пользователя «лишние» панели, что позволит детальнее осмотреть созданный макет;

– страница, содержащая меню, позволяющее пользователю загрузить новые пресеты, предоставляемые разработчиком продукта;

– страница компонентов, содержащая основные инструменты и функциональность;

– «песочница», предоставляющая пользователю возможность протестировать созданный макет приложения;

– страница, содержащая сгенерированный код макета, сохранить макет как пресет.

Навигацию между видами осуществляет меню-оболочка. Однако как ее подключение, так и подключение всех перечисленных выше страниц является опциональным и лишь предоставляет дополнительную функциональность, которая в будущем может быть, например, подвержена дополнительной монитезации разработком, интегрирующим данное программное средство в свой продукт.

Основной и обяательной страницей является страница компонентов, содержащая главыне инструменты:

– палитра компонентов, осущестляющая загрузку компонентов и пресетов и предоставляющая доступ к их спискам;

– инспектор свойств, содержащий меню доступа к свойствам компонентов, настройкам привязки и т.д.;

– грид-разметка, несущая основную функциональность, является ме-

стом создания макетов.

Расположение компонениов никак не влияет на функциональность, так что этот пункт целиком и полностью остается за разработчиком, интегрирующим данное программное средство в свой продукт.

Основное преимущество заключается в том, что количество одновременно доступных пользователю экзепляров не ограничивается функциональностью, однако есть одно обязательноек исполнению правило: один грид (грид-разметка)может связываться с каким угодно количеством палитр компонентов и инспекторов свойств, но каждый инспектор свойств и каждая палитра компонентов может быть «связана» лишь с одним гридом.

Предоставляемая гибкость программного средства дает пространство для разнообразия реализации разработку, например разработчик может реализовать коллаборативный сервис, который будет позволять в режиме реального времени нескольким людям «собирать» свое приложение.

4 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

Для реализации программного средства использовался Webix - JavaScript библиотека, разработанная специально для работы с UI. Инструменты, предоставляемые данной библиотекой, позволяют быстрее разрабатывать веб-приложения благодаря широкому набору готовых «умных» компонентов с богатым набором предоставляемых свойств и методов.

Основная функциональность программного средства заключена в трех главных компонентах:

– палитра компонентов - панель, где отображается список доступных компонентов и пресетов, доступных пользователю, и именно отсюда осуществляется перетягивание компонентов на грид;

– грид - разметка, куда осуществляется «бросание» компонентов пользователем;

– инспектор свойств - панель, созданная для редактирования свойств компонентов, содержит все необходимые инструменты, чтобы определить то, как будет работать компонент.

На рисунке 4.1 представлена диаграмма классов графических компонентов типа «layout», представленных в приложении. Список классов может пополняться в результате расширения приложения разработчиками.

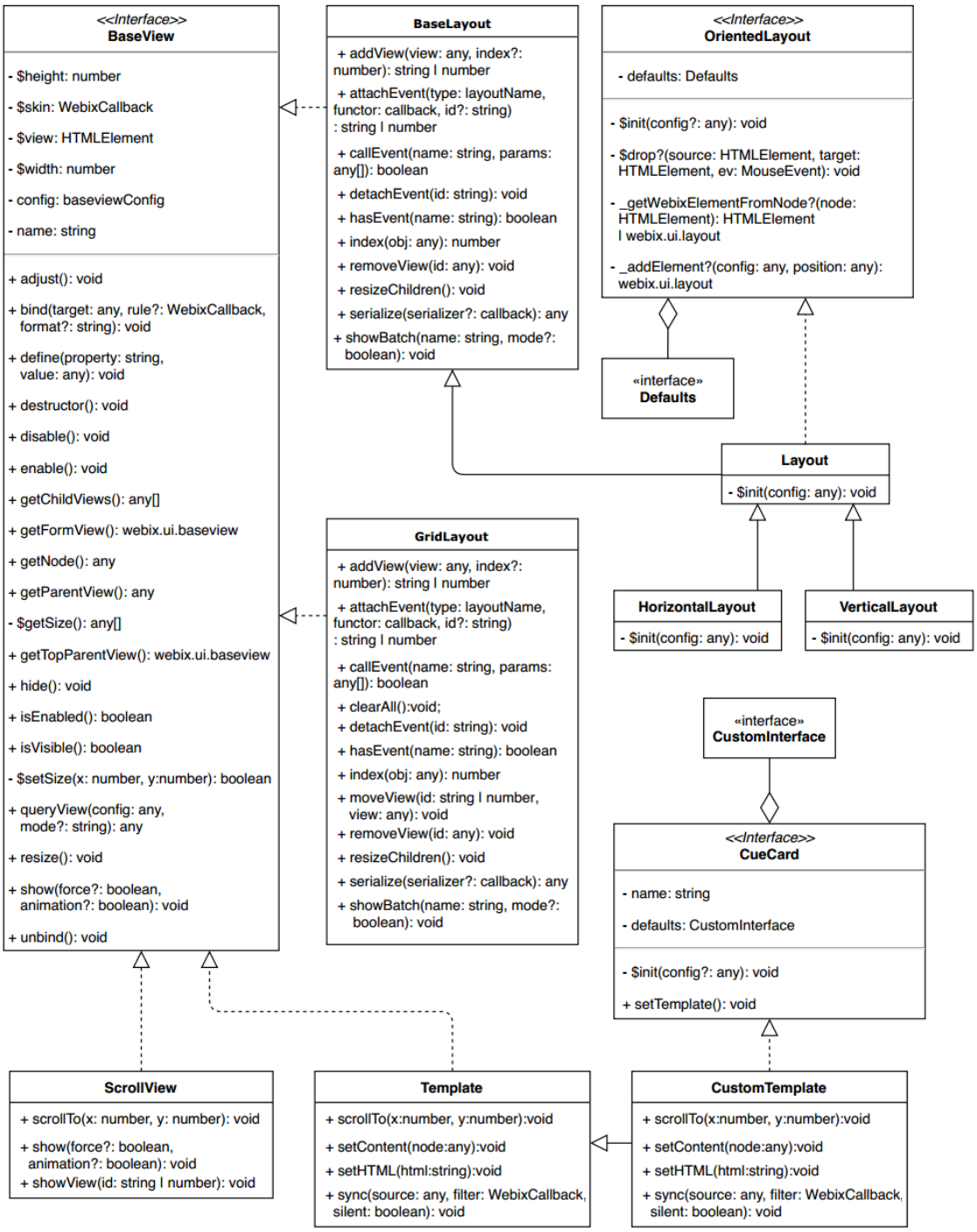

Рисунок 4.1 – Паттерн проектирования «Одиночка»

4.1 Сборщик проекта

Инструменты сборки стали неотъемлемой частью веб-разработки, в основном из-за возрастающей сложности JS-приложений. Бандлеры позволяют нам упаковывать, компилировать, организовывать множество ресурсов и библиотек, необходимых для современного веб-проекта.

В ходе процесса разработки в качестве сборщика проекта был использован Webpack.

Webpack - сборщик проекта, который помогает автоматизировать второстепенные задачи веб-разработки и чаще всего используется для оперирования файлами и папками внутри проекта, преобразования таблиц стилей, написанных с применением синтаксиса SASS, LESS и др. в формат стандартных таблиц стилей CSS. Его также часто используют для выполнения задачи минификации кода проекта при помощи специальных плагинов – оптимизации в один или несколько файлов, что исключительно положительно сказывается на конечной производительности проекта.

Помимо этого, Webpack может путем анализа кода проекта удалять из него неиспользуемые части, что может привести к значительному уменьшению минифицированного варианта проекта. Из всего вышесказанного можно сделать вывод о том, что Webpack – мощный инструмент, который существенно облегчает процесс разработки проекта.

Webpack, мощный бандлер с открытым исходным кодом, который может обрабатывать огромное количество различных задач и является одним из самых мощных и гибких инструментов для сборки frontend.

Плюсы:

– великолепен для работы с одностраничными приложениями;

– воспринимает как require()- так и import-синтаксисы модуля;

– позволяет осуществлять продвинутое разделение кода;

– hot reload для более быстрой разработки с помощью React, Vue.js и подобных фреймворков;

– наиболее популярный инструмент разработки по версии обзора JS в 2016 году.

Минусы:

– не подойдет для новичков;

– работа с файлами CSS, картинками и другими не JS ресурсами по началу сбивает с толку;

– очень много изменений, большинство гайдов 2016 уже устарели;

4.2 Среда разработки Visual Studio Code

Visual Studio Code — программное средство, разработанный Microsoft для Windows, Linux и macOS. Позиционируется как «лёгкий» редактор кода для кроссплатформенной разработки веб- и облачных приложений. Включает в себя отладчик, инструменты для работы с Git, подсветку синтаксиса, IntelliSense и средства для рефакторинга. Имеет широкие возможности для кастомизации: пользовательские темы, сочетания клавиш и файлы конфигурации.

Распространяется бесплатно, разрабатывается как программное обеспечение с открытым исходным кодом, но готовые сборки распространяются под проприетарной лицензией.

Visual Studio Code основан на Electron — фреймворк, позволяющий с использованием Node.js разрабатывать настольные приложения, которые работают на движке Blink. Несмотря на то, что редактор основан на Electron, он не использует редактор Atom. Вместо него реализуется веб-редактор Monaco,разработанный для Visual Studio Online.

Visual Studio Code — это редактор исходного кода. Он поддерживает ряд языков программирования, подсветку синтаксиса, IntelliSense, рефакторинг, отладку, навигацию по коду, поддержку Git и другие возможности. Многие возможности Visual Studio Code не доступны через графический интерфейс, зачастую они используются через палитру команд или JSON файлы (например, пользовательские настройки). Палитра команд представляет собой подобие командной строки, которая вызывается сочетанием клавиш.

Visual Studio также позволяет заменять кодовую страницу при сохранении документа, символы перевода строки и язык программирования текущего документа.

С 2018 года появилось расширение Python для Visual Studio Code с открытым исходным кодом. Оно предоставляет разработчикам широкие возможности для редактирования, отладки и тестирования кода.

На март 2019 года посредством встроенного в продукт пользовательского интерфейса можно загрузить и установить несколько тысяч расширений только в категории «programming languages» (языки программирования).

4.3 Описание структуры приложения

На рисунке 4.2 представлена файловая структура проекта, который является клиентским приложением.

Проект состоит из следующих основных частей:

– папка Components, содержащая исходный код компонентов;

– папка helpers, содержащая некторые часто применяемые участки кода, вынесенные в отдельные функции;

– папка styles, содержащая CSS-стили для всего проекта.

Папка Components в свою очередь включает в себя все существующие компоненты и состоит из:

– папка componentsPallet, содержащая исходный код палитры компонентов;

– папка componentsLayout, содержащая исходный код грида;

– папка cueCards, содержащая исходный код пользовательских шаблонов;

– папка layouts, содержащая исходный код основных предоставляемых компонентов;

– папка propertyInspector, содержащая исходный код инспектора свойств.

Папка layouts, как уже было сказано выше, включает в себя папки, содержащие исходный код каждого из присутствующих в программном компонентов. Этими папками являются:

– папка baselayout содержит исходный базового компонента, чья функциональность используется во всех компоненах типа «layout»;

– папка gridlayout, содержит исходный особого компонента типа «layout», который использует совершенно другой способ позиционирования дочерних элементов, нежели baselayout: в отличие от последнего, где используется смещение в пикселях от левой верхней точки, здесь используется абсолютное позиционирование по виртуальным клеткам;

– папка helpers, содержащая некторые часто применяемые в компонентах участки кода, вынесенные в отдельные функции;

– папки horizontal layout и vertical layout содержат исходный код одноименных компонентов, осуществояющих позиционирование дочерних компонентов либо в колонки (горизонтально), либо в строки (вертикально).

4.4 Разработка палитры компонентов

Для инициализации компонента палитры были использованы компоненты библиотеки Webix. Для отображения разных вкладок на панели использвался компонент TabView, а для отображения данных и источника «перетягивания» комопннетов использовался компонент DataView, так как имеет встроенные возможности для работы с алгоритмом Drag-n-drop.

4.4.1 Способ предсоставления компонентов

Каждой вкладке соответствует свой DataView, отображающий дозированные согласно названию вкладки данные, управляет отображением же TabView. Далее будет представлен основной код палитры компонентов.

```
import getAllTabConfig from './palletTabs/all';
import { getPresetsTabConfig } from './palletTabs/presetsTab';

const getDefaultComponentPalletLayout = dataContent => ({
```

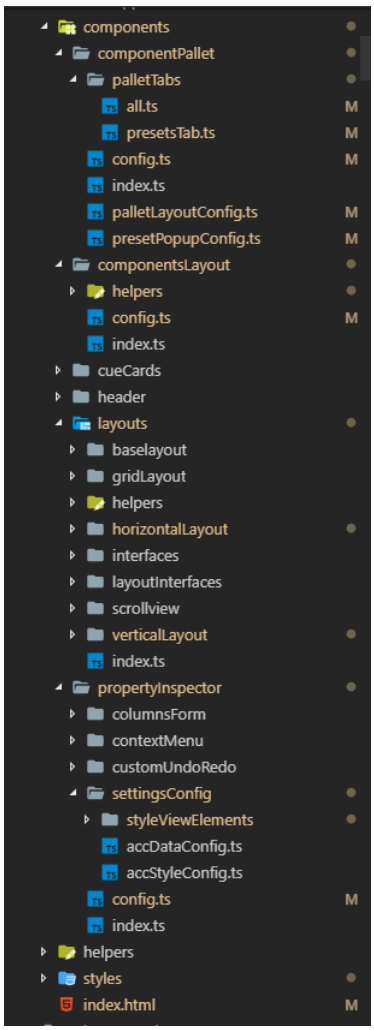

Рисунок 4.2 – Файловая структура проекта

```
rows: [
  {
    view: 'search',
    placeholder: 'Search 1,000,000 images...'
  },
  {
    view: 'tabview',
    cells: [
      {
        header: 'All',
        body: getAllTabConfig(dataContent)
      },
      {
        header: 'Presets',
        body: getPresetsTabConfig()
      },
      ...componentContainers(dataContent)
    ]
  }
```

```
  ]
});
```

Содержимое массива, возвращаемого функцией componentContainers являются DataView компоненты, каждый и который соответствует сожержанием своей вкладке.

```
const DATAVIEW_TEMPLATE = '<span class="webix-icon fa fa-#icon
   #"></span>  #element#';
const LAYOUT_DATAVIEW_CONFIG = 'pallet:dataview:layouts';
const COMPONENTS_DATAVIEW_CONFIG = 'pallet:dataview:components';
const CUECARDS_DATAVIEW_CONFIG = 'pallet:dataview:cuecards';

const componentContainers = data => [{
  header: 'Layouts',
  body: {
    view: 'dataview',
    template: DATAVIEW_TEMPLATE,
    id: LAYOUT_DATAVIEW_CONFIG,
    xCount: 3,
    drag: 'source',
    minHeight: 200,
    data: data.layouts
  }
},
{
  header: 'Components',
  body: {
    view: 'dataview',
    template: DATAVIEW_TEMPLATE,
    id: COMPONENTS_DATAVIEW_CONFIG,
    xCount: 3,
    drag: 'source',
    minHeight: 200,
    data: data.components
  }
},
{
  header: 'Cue cards',
  body: {
    view: 'dataview',
    template: DATAVIEW_TEMPLATE,
    id: CUECARDS_DATAVIEW_CONFIG,
    xCount: 3,
    drag: 'source',
    minHeight: 200,
    data: data.templates.map(element => ({
```

```
      ...element,
      template: element.template
    }))
  }
}]
```

Каждый DataView компонент отвечает за отображение своей части данных, а указание свойства «drag: 'source' предоставляет возможность использовать компонент как источник данных (пункта «тащи» или «drag») для реализации алгоритма Drag-n-drop.

Объект конфигураций самого компонента выглядит следующим образом:

```
import getDefaultComponentPalletLayout from './palletLayoutConfig
  ';
import { onPresetLoad } from './palletTabs/presetsTab';
import ComponentPallet from '../layouts/interfaces/
  ComponentPallet';

const componentPalletConfig: ComponentPallet = {
  name: 'AppOrchid-Component-Pallet',
  defaults: {
    // defaults here

  },
  $init() {
    // initialization here
  },
  _getComponentViewConfig() {
    if (!this.config.componentView || typeof this.config.
      componentView !== 'object') {
      return this.config.defaultComponentView;
    }

    return this.config.componentView;
  },
  _getComponentView() {
    return this.queryView({ ...this.config.defaultComponentView
      });
  }
};
```

Загрузка содержимого DataView компонентов - преопределенных конфигураций графических компонентов, осуществляется самим Component-Pallet при его инициализации путем, регламентированным библиотекой Webix.

```
$init() {
  this.$ready.push(
    () => {
    webix.ajax().get(this.config.widgets)
      .then(res => res.json())
      .then(res => ({
        layouts: res.layouts.map(item => ({ ...item, widgetType:
           'layout' }),
        components: res.components.map(item => ({ ...item,
           widgetType:'component' })),
        templates: res.templates.map(item => ({ ...item,
           widgetType: 'card'}))
      }))
      .then((res) => {
        this.addView({
          ...getDefaultComponentPalletLayout(res)
        });
      })
      .catch((e) => {
        console.error(e);
      });
    },
    () => {
      if (!this.config.gridId) { return; }
      let loadPreset = onPresetLoad.bind(this);
      this.attachEvent('onPresetLoad', loadPreset);
    }
  );
}
```

Применение кастомной конфигурации к базовому компоненту осуществляется путем применения метода protoUI глобального объекта webix к конфигурации нашего компонента Component-Pallet и базовым классам, предоставляемым библиотекой Webix:

- webix.IdSpace - для изоляции содержимого;
- webix.ui.layout - для доступа к функциональности базового лэйаута.

```
webix.protoUI(
  componentPalletConfig,
  webix.IdSpace,
  webix.ui.layout
);
```

По умолчанию предполагается, что в получаемом в результате запроса JavaScript-объекте должны быть поля: layouts, components и card - три

вида лэйаутов. Однако, эти параметры могут быть изменены разработчиком, интегрирующим данное програмнное средство в свой проект.

4.4.2 Пресеты

В палитре компонентов, как указывалось ранее, доступна возможность загрузки и применения пресетов - предопределенных наборов компонентов, которые были созданы и предоставлены разработчиками или, например, другими пользователями.

Доступ к пресетам осуществляется через соответствующую вкладку «Presets». Лист пресетов получает их конфигурации из объекта, реализующего доступ к данным в библиотеке Webix - DataCollection.

```
{
  id: getPresetsListId(),
  borderless: true,
  data: webix.storage.local.get('collection') || [{ name: 'No
     presetsfound' }],
  view: 'list',
  select: true,
  template: `#name#${DELETE_PRESET_ICON_TEMPLATE}`,
  on: {
      onItemDblClick(id): void {
      const topParent = this.getTopParentView();
      topParent.callEvent('onPresetLoad', [this.getItem(id)]);
      }
  },
  onClick: {
    'fa-trash-o': function (ev, id): void {
      const self = this;
      webix.confirm({
        text: 'Item data will be lost. <br/> Are you sure?',
        ok: 'Yes',
        cancel: 'Cancel',
        callback: (res) => {
          removePreset.apply(self, [res, id]);
        }
      });
    }
  }
}
```

Сохранение текущей конфигурации осуществляется путем сериализации содержимого грида. Для этого вызывается метод объекта грида getSerializedCollection, логика которого будет рассмотрена позднее.

```
const topParent = $$(this.getTopParentView().config.master)
.getTopParentView();

const gridId = $$((topParent.config).gridId);

const newElement = savePreset(this.getValue(), $$(gridId)
  .getSerializedCollection());
this.getTopParentView().hide();

if (newElement) {
  topParent.$$(getPresetsListId()).parse(newElement);
}
```

Применение пресета из списка доступных осуществляется в несколько этапов. Первым является вызов события onPresetLoad, на срабатывание которого вызывается колбэк с соответствующим именем, в котором осуществляется первичная проверка данных и дальнейшая делегация выполнения применения пресета.

```
function onPresetLoad(preset) {
  const grid = ($$(this.config.gridId));

  if (preset.content) {
    let arr = $$(this.config.gridId)
      .getChildViews();

    for (let i = arr.length - 1; i > 0; i--) {
      grid.removeView(arr[i]);
    }

    webix.delay(() => {
      loadPreset(preset.content, grid);
    });

    webix.message({
      text: `Preset ${preset.name} loaded`,
      expire: 1000
    });
  } else {
    webix.message({
      text: 'No data provided',
      type: 'error'
    });
  }
}
```

Вызываемая внутри функция loadPreset вызывает функцию

saveAndRender для каждого объекта, которому соответствует компонент.

```
const loadPreset = (collection: object, grid: ComponentsLayout)=>
    {
  Object.keys(collection)
    .forEach((presetId) => {
    saveAndRender(collection[presetId], grid);
  });
};
```

Метод saveAndRender в свою очередь добавляет нужные позиционные свойства компонентам, которые будут помещаться на грид и вызывает у грида метод рендера. В качестве параметра туда передается конфиг компонента и его будущие координаты.

```
const saveAndRender = (source: any, grid: ComponentsLayout) => {
  const id = webix.uid();
  let item = { id, ...source };

  if (!item.parentId) {
    const width = item.width || 0;
    const height = item.height || 0;

    const offsetX = Number(item.left + (width / 2));
    const offsetY = Number(item.top + (height / 2));

    grid._addMovableElement(item, {
      offsetX,
      offsetY
    });
  }
  return item;
};
```

Удаление пресета из списка осуществляется путем удаления пресета из коллекции, к которой «привязан» список пресетов. Путем простой фильтрации по условию неравенства уникальному свойству удаляемого пресета и осуществляется процесс удаления.

```
function removePreset(result: any, id: string | number) {
  if (result) {
    let presetsCollection = webix.storage.local.get('collection')
      ;
    if (!presetsCollection || !presetsCollection.length) {
```

```
      return;
    }
    presetsCollection.filter(element => element.name !== this.
       getItem(id).name);

    this.remove(id);

    webix.storage.local.put('collection', presetsCollection);
  }
}
```

Резюмируя, можно сделать вывод, что был разработан гибкий инструмент для использования предопределенных наборов компонентов, предоставляющий функциональность как добавления и применения пресетов, так и их удаления.

4.5 Разработка грида

Грид выступает инструментом для отображения компонентов, перетянутых с палитры компонентов. Это основной компонент, занимающий большую часть экрана. Он представляет собой разметку с сеткой, размер которой может быть изменен.

4.5.1 Алгоритм Drag-n-drop

Создание и перемещение компонентов осуществляется методами, реализующими алгоритм Drag-n-drop. Для этого был разработан объект Movable, методы которого предоставлет методы $dragCreate, $dragPos и $dragDestroy.

Метод $dragCreate отвечает за начало перетягивания компонента. Библиотека Webix предполагает наличие метода с таким названием и выполнение им строго определенных функций, в частности, если начинает перетягиватся компонент, возвращать его DOM-элемент, иначе - false.

```
$dragCreate(object, e) {
  let master = webix.DragControl.getMaster(object).master;

  if ($$(e.target.getAttribute('view_id')) &&
    $$(e.target.getAttribute('view_id')).config.view === 'resizer
       ') {
    return false;
  }

  if (master.config.move) {
    let offset = webix.html.offset(object);
```

```
    let pos = webix.html.pos(e);

    webix.DragControl.top = offset.y - pos.y;
    webix.DragControl.left = offset.x - pos.x;
    return webix.toNode(master.$view);
  }
  return false;
}
```

Метод $dragDestroy осуществляет финализацию процесса перетягивания компонента: очистку используемых ресурсов, убирание тени от движения компонента, вызов события окончания движения и т.д.

```
$dragDestroy() {
  const master = this.master.getChildViews()[0.$view.
     getBoundingClientRect();
  const grid = this.master.getParentView();

  const gridBounds = this.master.getParentView()
    .$view
    .getBoundingClientRect();
  let view = this.master;

  let diffX = master.x - gridBounds.x;
  let diffY = master.y - gridBounds.y;

  let { left, top } = stickToGrid(this.master,diffX, diffY, grid.
     config.separation);

  if (view._settings) {
    view._settings.top = top + 7;
    view._settings.left = left + 7;
  }

  webix.DragControl.top = 5;
  webix.DragControl.left = 5;

  view.resize();

  removeShadow();

  this.master.callEvent('onViewMoveEnd', []);
  let item = this.master.getChildViews()[0];

  item.config.top = this.master.config.top;
  item.config.left = this.master.config.left;
```

```
  const elem = view.getChildViews()[0];

  elem.callEvent('onBlur', []);
  elem.callEvent('onFocus', [elem]);
}
```

Метод $dragPos отвечает за перерисовку компонента и грида во время движения первого.

```
$dragPos(pos, e) {
  this.master.callEvent('onViewMove', [pos, e]);

  const grid = this.master.getParentView();
  const gridBounds = grid.$view.getBoundingClientRec();

  let context = webix.DragControl.getContext();
  let control = context.source;

  pos.x = e.clientX - gridBounds.x;
  control.style.left = pos.x;
  pos.y = e.clientY - gridBounds.y;
  control.style.top = pos.y;

  const master = this.master.getChildViews()[0.$view.
     getBoundingClientRect();

  let diffX = master.left - gridBounds.x;
  let diffY = master.top - gridBounds.y;

  let { left, top } = stickToGrid(this.master, diffX, diffY, grid
     .config.separation);

  let shadowRect = document.getElementsByClassNam('element-shadow
     ')[0];

  if (shadowRect) {
    removeShadow();
  }

  if (!shadowRect
    || (shadowRect.getBoundingClientRect().left !== left
    && shadowRect.getBoundingClientRect().top !== top)) {
      drawShadow(this.master.$view, left + 10, top + 10, grid.
         getNode());
  }
}
```

4.5.2 Разработка сетки грида и реализация «магнитной привязки»

Сетка грида выполняет не только декоративную функцию и несколько упрощает визуальное выравнивание компонентов, но и обладает свойством магнитной привязки. Алгоритм отрисовки сетки представлен ниже.

```
drawCanvasGrid() {
  const self = this;
  const conf = {
    template: () => `<canvas
      id ="${self._canvasId}"
      height="${document.body.clientHeight}"
      width="${document.body.clientWidth}"
    />`,

    top: 0,
    left: 0,
    zIndex: -10,

    autoheight: true,
    presetParse: false,

    on: {
        onAfterRender() {
            this.define('height', document.body.clientHeight);
            this.define('width', document.body.clientWidth);
            this.resize();

            self._drawGrid();
        }
    }
  };
  this.addView(conf);
}
```

Данный алгоритм предоставлет возможность редактировать размер сетки, суть которой заключается в том, что компоненты можно передвигать лишь с шагом, равным размеру клетки в пикселях. Вызовы метода stickToGrid выполняют дополнительные высчитывания координат места, куда будет позиционирован перетягиваемый компонент по окончании.

Также помимо сетки, присутствует тень от компонента, которая показывает, куда компонент будет помещен, когда пользователь отпустит нажатую для перетягивания компонента левую кнопку мыши или уберет палец от сенсорного экрана.

Алгоритм рисования тени предполагает вызов перерисовки тени на

каждое событие движения мыши.

```
const drawShadow = (htmlView, left, top, grid) => {
  const shadow = document.createElement('div');

  const viewRect = htmlView.children[0.getBoundingClientRect();
  shadow.className = 'element-shadow';

  shadow.style.top = `${top}px`;
  shadow.style.left = `${left}px`;
  shadow.style.width = `${viewRect.width - 1}px`;
  shadow.style.height = `${viewRect.height - 1}px`;
  shadow.style.position = 'relative';

  grid.appendChild(shadow);
}

const removeShadowGrid = () => {
  let div = document.getElementsByClassName('grid-shadow')[0];
  div.parentNode.removeChild(div);
};
```

Добавление функциональности Drag-n-drop осуществляется путем вызова метода addDrop объекта DragControl, который доступен через глобальный объект webix. Вызов метода происходит в методе инициализации грида $init.

```
webix.DragControl.addDrop(self.$view, {
  $drop(source, target, ev) {
    let dnd = webix.DragControl.getContext();

    if (dnd.from.name === 'dataview' &&
      (dnd.from.getItem(dnd.source[0]).widgetType === 'layout' ||
         dnd.from.getItem(dnd.source[0]).widgetType === 'card')
    ) {
        if (dnd.from.name === 'dataview') {
          let elementConfig = webix.copy(dnd.from.getItem(dnd.
             source[0].config);

          if (elementConfig.view === 'AppOrchid-Gridlayout') {
            self._onGridLayoutDrop(elementConfig, ev);
            return;
          }

          elementConfig.id = webix.uid();
          self._addMovableElement(elementConfig, ev);
```

```
        }
    } else if (dnd.from.name === 'dataview' &&
    dnd.from.getItem(dnd.source[0]).widgetType !== 'layout') {
      webix.message({
        text: 'Only layouts can be placed on the grid',
        type: 'error',
        expire: 2000
      });
    }
  }
})
```

Добавление элемента на грид осуществляется методом внутренним приватным методом _addMovableElement, который добавляет внутренние свойства к конфигу компонента, необходимые для дальнейшей работы.

```
_addMovableElement(config, ev) {
  let height;
  let width;
  let top;
  let left;
  const self = this;

  // when vertical or horizontal layout
  if (!config.width || !config.height) {
    width = this.config.separation;
    height = this.config.separation;
  } else {
    width = config.width;
    height = config.height;
  }

  top = ev ? ev.offsetY - (height / 2) : DEFAULT_OFFSET;
  left = ev ? ev.offsetX - (width / 2) : DEFAULT_OFFSET;

  config.top = top;
  config.left = left;

  function _recursiveMovableEvents(element) {
    if (element.cols && element.cols.length) {
      element.cols = element.cols.map(subElement =>
        _recursiveMovableEvents(subElement));
    }

    if (element.rows && element.rows.length) {
      element.rows = element.rows.map(subElement =>
        _recursiveMovableEvents(subElement));
```

```
    }

    if (element.cells && element.cells.length) {
      element.cells = element.cells.map(subElement =>
        _recursiveMovableEvents(subElement));
    }

    return Object.assign(
      {},
      self._getMovableElementEvents(element),
      webix.copy(element)
    );
  }

  let baseElement = {
    view: 'elementMovable',
    move: true,
    presetParse: false,

    css: {
      border: '3px solid transparent'
    },
    rows: [
      _recursiveMovableEvents(config)
    ],

    id: webix.uid(),
    top,
    left
  };

  this.addView(baseElement);
  this.callEvent('componentAdded', [baseElement.id]);
  return config;
}
```

Метод также добавляет необходимые обработчики событий как самому компоненту,так и всем его дочерним, что позволяет быть уверенным, что компоненты будут вести себя одинакого в разных сценариях использования.

Весь алгоритм в целом и некоторые его аспекты показаны на рисунке 4.3

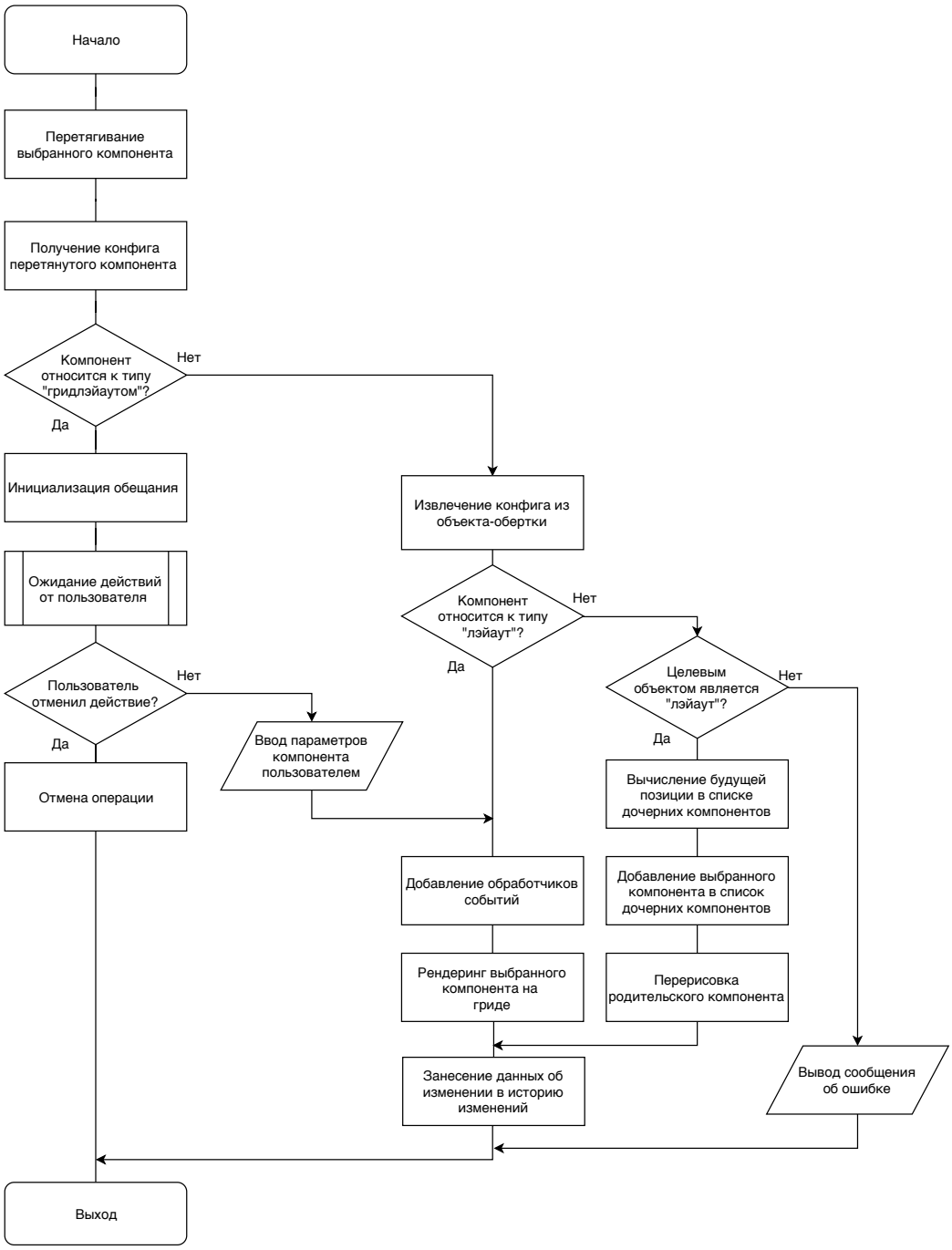

Рисунок 4.3 – Алгоритм компоновки графических компонентов

4.6 Разработка инспектора свойств

Инспектор свойств является панелью, содержащей множество вкладок, поэтому были использованы такие базовые графические компоненты, предоставляемые библиотекой Webix, как:

– TabView - для дозирования количества одновременно отображаемой информации путем предоставления доступа к ее частям посредством разделения на страницы;

– Accordion - для группирования отображаемой информации в логически или технически обусловленные группы свойств.

Конфиг компонента представляет собой набор «страниц», реализованный посредством компонента TabView. Каждая страница содержит в себе компонент Accordion, позволяющий манипулировать отображением собственного содержимого.

```
this.addView({
  rows: [
    undoRedoConfig,
    { ...getSearchPanel() },
    {
      id: webix.uid(),
      borderless: true,
      view: 'tabview',
      cells: [
        {
          header: 'Data',
          ...getAccDataConfig()
        }
        {
          header: 'Style',
          ...getAccStyleConfig(this)
        }
        {
          header: 'Code',
          ...getElementCodeConfig()
        }
      ]
    }
  ]
})
```

Компонент Accordion, использующийся с целью отображения свойств выделенных компонентов, в свою очередь использует специальный компонент библиотеки Webix, называющийся «property», который был специально создан для подобных целей.

Метод onAfterEditStop вызывается по окончании пользователем ввода данных в поле свойства. Здесь осуществляются проверки введенных данных на валидность и их дальнейшая обработка.

На случай ввода неверных данных предусмотрены дополнительные обработчики и средства вывода информации, с целью уведомить пользователя об ошибке.

```
{
  id: getSettingsId(),
  view: 'property',
  onContext: {},
  elements: getElements(),
  autoheight: true,
  nameWidth: 120,
  on: {
    onAfterEditStop(state, editor): void {
      if (Number.isInteger(Number(state.value))) {
        state.value = Number(state.value);
      }

      if (Number(state.value) !== state.old) {
        let view = <any>this.getTopParentView()
          .getLastActiveComponent();

        if (view) {
          if (editor.id === 'url') {
            view.clearAll();
            view.config.data = null;
          } else if (editor.id === 'top' || editor.id === 'left')
              {
            view.getParentView().config[editor.id] = state.value;
            view.getParentView()
                .resize();
          }

          // visible component editing
          view.config[editor.id] = state.value;

          if (view.setTemplate) {
            view.setTemplate();
          }
          if (view.refresh) {
            view.refresh();
          }

          view.resize();
          view.callEvent('onBlur');
          view.callEvent('onFocus', [view]);
        } else {
          webix.message({
            text: 'Attempt to change property of not existing
               item',
            type: 'error',
            expire: 2000
```

```
          });
        }
      }
    }
  }
}
```

Основным инструментом для работы с компонентами является панель свойств, отображающая свойства выделенного компонента.

```
definePropertyTable(viewConfig) {
  let elements = [];
  let propertyTable = this.$$(getSettingsId());

  this.$$(getAccStyleId()).config.data = viewConfig;

  const elementCodeView = this.$$(getElementCodeId());

  if (viewConfig) {
      if (viewConfig.view === 'datatable') {
          elements = getDatatableElements();
      } else {
        Object.entries(viewConfig)
          .forEach(([key]) => {
            if (
              key === 'id'
              || key === '_inner'
              || key === 'template'
            ) {
              return;
            }

            let type = 'text';
            if (key.search(/color/gi) !== -1) {
              type = 'color';
            }

            elements.push({
              id: key,
              label: key,
              type
            });
        });
      }
  }

  elementCodeView.setValue(`${getPrettyJson(viewConfig)}`);
```

```
  elementCodeView.refresh();

  propertyTable.define('elements', elements);

  let parsedItem = getParsedConfig(viewConfig);

  propertyTable.setValues(parsedItem);

  propertyTable.resize();
}
```

Чтобы после выделения пользователем компонента его свойства отображались во вкладке свойства инспектора свойств необходимо было добавить в его метод инициализации соовтествующий обработчик события.

```
() => {
  if (!this.config.gridId) { return; }
  eventHandlerInit(this.config.gridId);
  const grid = $$(this.config.gridId);

  grid.attachEvent('onMovableElementSelect', target) =>{
    focusEventHandler(target);
    this.setLastActiveComponent(target);
    let configCopy = null;

    try {
      configCopy = webix.copy(target.config);
    } catch (e) {
      configCopy = webix.clone(target.config);
    }
    this.definePropertyTable(configCopy);

    if (target.config.view === 'datatable') {
      showColumnsForm(this, target);
    } else {
      hideColumnsForm(this);
    }
  });
}
```

При срабатывании события onMovableElementSelect происходит копирование свойств компонента с помощью метода библиотеки. Функции showColumnsForm и hideColumnsForm отвечают за управление отображением дополнительной вкладки свойств, доступной лишь после выделения особого компонента, предоставляемого библиотекой Webix, нужнающегося в дополнительной конфигурации - Datatable.

Данная особенность обусловлена особой струкурой компонента и соответсвующего ему DOM-элемента.

Помимо всего вышеперечисленного, инспектор свойств предоставляет возможность поиска свойства как по ключу, так и по значению, что значительно упрощает работу с комплексными компонентами, содержащими большое количесво свойств.

Для этого в самом верху инспектора свойств расположена панель ввода с соовтествующей иконкой.

Далее будет представлен код, описывающий логику данного компонента.

```
const tabviewSearch = {
  id: getSearchPanelId(),
  view: 'search',
  hotkey: 'enter',
  placeholder: 'Search any property',
  on: {
    onChange() {
      getInputValue.call(this);
    },
    onFocus() {
      // blocking events in order to prevent starting search
      this.blockEvent();
      this.getNode().children[0].children[0].select();
      this.unblockEvent();
    },
    onEnter() {
      getInputValue.call(this);
    }
  }
};
```

Секция on содержит обработчики одноименных событий. Свойство hotkey указывает горячую клавишу для вызова события onEnter, которое осуществляет вызов функции getInputValue с контектом объекта компонента поиска.

Функция getInputValue инкапсулирует функциональность поиска данных по ключу и значению. Ее код преставлен ниже.

```
function getInputValue(): void {
  const list = <webix.ui.list>this.getTopParentView().$$(
     getSearchListId());
  const val: string = this.getTopParentView().$$(getSearchPanelId
     ()).getValue().toLowerCase()
```

```
    .trim();
  const store: object = getParsedConfig(this.getTopParentView().
     $$(getAccStyleId()).config.data);

  list.clearAll();

  if (store && val) {
    this.getTopParentView().$$(getSearchListId()).clearAll();

    Object.entries(store).forEach(([key, value]) => {
      if (JSON.stringify(value) !== undefined
        && (key.toLowerCase().indexOf(val) !== -1
          || JSON.stringify(value).indexOf(val) !== -1)
      ) {
        if (typeof value === 'string') {
          value = value.trim();
        }
        list.parse({ key, value }, '');
      }
    });
  }

  this.blur();
  list.getParentView().getParentView().show();
  list.getParentView().define({ collapsed: false });
  list.getParentView().refresh();
}
```

По окончании выполнения вышеуказанных в функции getInputValue действий, все найденные свойства добавятся в содержимое компонента, отображающего результаты поиска в виде списка. Вкладка с результатами поиска будет открыта автоматически, что определенно благоприятно скажется на UX.

Функциональность самого списка результатов поиска не ограничивается одним лишь отображением самих результатов поиска - при нажатии на один из отображаемых результатов система автоматически отобразит пользователю нужную вкладку и, более того, сразу откроет поле с искомым свойтсвом в режиме редактирования.

Ниже будет предоставлен фрагмент кода с конфигом данного компонента, где будут видны некоторые детали реализации.

```
const searchListId = 'AC:settings:searchList';

const getSearchListId = (): string => searchListId;
```

```
const searchListConfig = {
  id: getSearchListId(),
  template: '{#key#: #value#}',
  select: true,
  view: 'list',
  on: {
    onItemClick(itemId): void {
      const accordion = this.getTopParentView().$$(
         getScrollViewId()).getParentView().getParentView();
      accordion.define('collapsed', false);

      let key = this.getItem(itemId).key;

      if (key !== 'id') {
        this.getTopParentView().$$(getSettingsId()).edit(this.
           getItem(itemId).key);
      }
    }
  },
  data: []
}
```

Подобная реализация также определенно благоприятно скажется на UX, ведь в перспективе пользователю необходимо будет совершать значительно меньше действий для достижения желаемого.

5 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПС

В данном разделе проведем динамическое ручное тестирование. В таблице 5.1 приведен список тестовых случаев, относящихся к позитивному тестированию.

Тестирование производилось при версиях зависимостей, указанных ниже. Ввиду того, что некоторые зависимости могли устареть, а некоторые их функции - перестать поддерживаться разработчиками пакета, стабильная работа на более поздних версиях пакетов не гарантируется.

```
{
    "dependencies": {
      "body-parser": "^1.18.2",
      "copy-webpack-plugin": "^4.5.1",
      "css-loader": "^0.28.11",
      "express": "^4.16.3",
      "express-fileupload": "^0.4.0",
      "file-loader": "^1.1.11",
      "html-webpack-plugin": "^3.1.0",
      "less": "^3.8.1",
      "less-loader": "^4.1.0",
      "style-loader": "^0.20.3",
      "ts-loader": "^4.5.0",
      "tslint": "^5.11.0",
      "typescript": "^3.0.1",
      "url-loader": "^1.0.1",
      "webpack-dev-server": "^3.1.5"
    },
    "devDependencies": {
      "babel-core": "^6.26.0",
      "babel-loader": "^7.1.4",
      "babel-plugin-module-resolver": "^3.1.1",
      "babel-plugin-transform-object-rest-spread": "^6.26.0",
      "babel-polyfill": "^6.26.0",
      "babel-preset-env": "^1.6.1",
      "eslint": "^4.19.1",
      "eslint-plugin-import": "^2.9.0",
      "extract-text-webpack-plugin": "^4.0.0-beta.0",
      "resolve-url-loader": "^2.3.0",
      "webpack": "^4.16.5",
      "webpack-cli": "^2.0.13"
    }
}
```

Таблица 5.1 – Тестовые случаи разрабатываемого программного средства

№	Описание тестового случая	Ожидаемый результат	Полученный результат
1	**Перетягивание обычного компонента на грид**. 1) зажать левую кнопку мыши на компоненте, не относящемся к типу «layout»; 2) перенянуть на грид. 3) отпустить левую кнопку мыши.	Компонент не отрисовывается на гриде. Отображается сообщение об ошибке, что компонент не относится к типу «layout».	Пройдено
2	**Выделение лэйаута**. 1) перетянуть «Horizontal layout» на грид; 2) левой кнопкой мыши нажать на созданный «Horizontal layout»,.	Границы компонента окрашиваются прерывистой зеленой линией, и на границе появляются точки для изменения размеров компонента.	Пройдено
3	**Добавление компонента в лэйаут**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Combo» в созданный «Horizontal layout»;	Компонент «Combo» будет отрисован внутри созданного «Horizontal layout».	Пройдено
4	**Выделение компонента внутри лэйаута**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Combo» в созданный «Horizontal layout»; 3) левой кнопкой мыши нажать на созданный «Combo».	Границы компонента окрашиваются прерывистой зеленой линией, а на границе не появляются точки для изменения размеров компонента.	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
5	**Редактирование компонента**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Combo» в созданный «Horizontal layout»; 3) левой кнопкой мыши нажать на созданный «Combo»; 4) открыть вкладку «STYLE» в инспекторе свойств; 5) левой кнопкой мыши нажать на на значение поля «width»; 6) используя только цифры, ввести данные о новых размерах. 7) левой кнопкой мыши нажать на кнопку «Enter».	Компонент перерисуется в соответствии с введенными данными.	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
6	**Редактирование лэйаута**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Combo» в созданный «Horizontal layout»; 3) выделить созданный «Horizontal layout»; 4) открыть вкладку «STYLE» в инспекторе свойств; 5) левой кнопкой мыши нажать на значение поля «width» или «height»; 6) используя только цифры, ввести данные о новых размерах; 7) левой кнопкой мыши нажать кнопку «Enter».	Лэйаут и его дочерние элементы перерисовываются в соответствии с введенными данными.	Пройдено
7	**Добавление компонента в конец непустого лэйаута**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Combo» в созданный «Horizontal layout»; 3) перетянуть «Datatable» на созданный «Horizontal layout».	Компонент «Datatable» будет отрисован крайним справа внутри созданного «Horizontal layout».	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
8	**Добавление компонента перед определенным компонентов внутри непустого лэйаута**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Combo» в созданный «Horizontal layout»; 3) перетянуть «Datatable» на созданный «Combo».	Компонент «Datatable» будет отрисован перед компонентом «Combo» внутри созданного «Horizontal layout».	Пройдено
9	**Удаление компонента внутри лэйаута с одним дочерним компонентом**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Combo» в созданный «Horizontal layout»; 3) левой кнопкой мыши нажать на созданный «Combo»; 4) открыть вкладку «STYLE» в инспекторе свойств; 5) левой кнопкой мыши нажать на кнопку «DELETE COMPONENT».	Компонент «Combo» будет удален внутри созданного «Horizontal layout» Сам лэйаут останется на гриде.	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
10	**Удаление непустого лэйаута**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Combo» в созданный «Horizontal layout»; 3) левой кнопкой мыши нажать на созданный «Horizontal layout»; 4) открыть вкладку «STYLE» в инспекторе свойств; 5) левой кнопкой мыши нажать на кнопку «DELETE COMPONENT».	Компонент «Horizontal layout» будет удален с грида вместе с дочерним элементом «Combo».	Пройдено
11	**Многоуровневая компоновка**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Datatable» в созданный «Horizontal layout»; 3) перенянуть «Vertical layout» в созданный «Horizontal layout»; 4) перенянуть «Combo» в созданный «Vertical layout»; 5) перенянуть «Datepicker» в созданный «Vertical layout».	Отображается компонент, состоящий из двух колонок: в первой «Datatable», во второй в строки расположены: «Combo» и «Datepicker».	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
12	**Удаление первого компонента среди дочерних компонентов лэйаута**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Datatable» в созданный «Horizontal layout»; 3) перенянуть «Vertical layout» в созданный «Horizontal layout»; 4) перенянуть «Combo» в созданный «Vertical layout»; 5) перенянуть «Datepicker» в созданный «Vertical layout»; 6) левой кнопкой мыши нажать на «Datatable»; 7) открыть вкладку «STYLE» в инспекторе свойств; 8) левой кнопкой мыши нажать на кнопку «DELETE COMPONENT».	Родительский <Horizontal layout» перерисовывается, компонент «Datatable» удалятся, и «Vertical layout» с компонентами «Combo» и «Datepicker» занимает его место.	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
13	**Появление тени от лэйаута на гриде**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Datepicker» на созданный «Horizontal layout»; 3) зажать левую кнопку мыши на созданном «Horizontal layout»; 4) перетянуть лэйаут с зажатой левой кнопкой мыши в другое место грида.	Сзади компонента появляется «тень».	Пройдено
14	**Магнитная сетка**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Datepicker» на созданный «Horizontal layout»; 3) зажать левую кнопку мыши на созданном «Horizontal layout»; 4) медленно перегивать лэйаут с зажатой левой кнопкой мыши; 5) отпустить левую кнопку мыши.	Сзади компонента появляется «тень», двигающаяся по гриду с шагом, равным размеру ребра клетки грида. После отпускания левой кнопки мыши компонент помещен туда, куда падала его тень.	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
15	**Изменение позиции созданного непустого лэйаута**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Datepicker» на созданный «Horizontal layout»; 3) зажать левую кнопку мыши на созданном «Horizontal layout»; 4) перетянуть лэйаут в другое место грида; 5) отпустить левую кнопку мыши.	Созданный на гриде «Horizontal layout» передвинут вмете с содержимым с начального положения.	Пройдено
16	**Редактирование уникальных свойств компонентов**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Leaf» на созданный «Horizontal layout»; 3) левой кнопкой мыши нажать на созданный компонент «Leaf»; 4) открыть вкладку «STYLE» в инспекторе свойств; 5) левой кнопкой мыши нажать на значение свойства «color»; 6) левой кнопкой мыши нажать на нужный цвет в выпадающем колопикере.	В созданном компоненте «Leaf» изменится цвет шаблона.	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
17	**Редактирование уникальных свойств компонентов**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Leaf» на созданный «Horizontal layout»; 3) левой кнопкой мыши нажать на созданный компонент «Chart»; 4) открыть вкладку «STYLE» в инспекторе свойств; 5) левой кнопкой мыши нажать на значение свойства «type»; 6) ввести строку «pie»; 7) левой кнопкой мыши нажать кнопку «Enter»; 8) ввести строку «line»; 9) левой кнопкой мыши нажать кнопку «Enter».	В созданном компоненте «Chart» меняется тип графика сначала на «pie», затем на «line». Проверка соответсвия вида графика осуществляется согласно документации Webix.	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
18	**Отображение дополнительных свойств компонента «Datatable»**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Datatable» на созданный «Horizontal layout»; 3) левой кнопкой мыши нажать на созданный компонент «Datatable»; 4) открыть вкладку «STYLE» в инспекторе свойств.	Во вкладке со свойствами появляется дополнительная вкладка для колонок компонента «Datatable».	Пройдено
19	**Отображение кода выделенного компонента при выборе вкладки «CODE»**. 1) перетянуть «Horizontal layout» на грид; 2) открыть вкладку «CODE» в инспекторе свойств.	Содержимое инспектора инструментов перерисуется, появится содержимое вкладки «CODE»: textarea с JavaScript-кодом объекта компонента «Horizontal layout».	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
20	**Переключение вкладок палитры компонентов**. 1) нажать на троеточие рядом со вкладкой «All» палитры компонентов; 2) в появившемся попапе нажать на «Components»; 3) нажать на троеточие рядом со вкладкой «Components» палитры компонентов; 4) в появившемся попапе левой кнопкой мыши нажать на «Layouts»; 5) нажать на троеточие рядом со вкладкой «Layouts» палитры компонентов; 6) в появившемся попапе левой кнопкой мыши нажать на «Custom templates»;	Происходит перерисовка содержимого списка палитры компонентов в соответствии с выбранной вкладкой.	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
21	**Сохранение нового пресета**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Datatable» на созданный «Horizontal layout»; 3) нажать на троеточие рядом со вкладкой «All» палитры компонентов; 4) в появившемся попапе нажать на «Presets»; 5) левой кнопкой мыши нажать на кнопку «SAVE CURRENT CONFIGURATION»; 6) ввести «123» в появившемся попапе; 7) левой кнопкой мыши нажать на кнопку «ADD».	В список пресетов в самый низ добавляется запись с именем «123».	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
22	**Создание нового пресета с пустым именем**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Datatable» на созданный «Horizontal layout»; 3) левой кнопкой мыши нажать на троеточие рядом со вкладкой «All» палитры компонентов; 4) в появившемся попапе нажать на «Presets»; 5) левой кнопкой мыши нажать на кнопку «SAVE CURRENT CONFIGURATION»; 6) левой кнопкой мыши нажать на кнопку «ADD».	В правом верхнем углу экрана появится сообщение об ошибке с текстом «Preset name can‘t be empty».	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
23	**Создание нового пресета с именем уже существующего пресета**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Datatable» на созданный «Horizontal layout»; 3) нажать на троеточие рядом со вкладкой «All» палитры компонентов; 4) в появившемся попапе нажать на «Presets»; 5) левой кнопкой мыши нажать на кнопку «SAVE CURRENT CONFIGURATION»; 6) ввести «123» в появившемся попапе; 7) левой кнопкой мыши нажать на кнопку «ADD»; 8) левой кнопкой мыши нажать на кнопку «SAVE CURRENT CONFIGURATION»; 9) повторить шаги 7 и 8.	В правом верхнем углу экрана появляется сообщение об ошибке с текстом «Item with this name already exists», а в список пресетов не добавляется новый пресет с введенным именем.	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
24	**Загрузка пресета**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Datatable» на созданный «Horizontal layout»; 3) нажать на троеточие рядом со вкладкой «All» палитры компонентов; 4) в появившемся попапе нажать на «Presets»; 5) левой кнопкой мыши нажать на кнопку «SAVE CURRENT CONFIGURATION»; 6) ввести «123» в появившемся попапе; 7) левой кнопкой мыши нажать на кнопку «ADD»; 8) левой кнопкой мыши дважды нажать на появившуся в списке надпись с текстом «123».	Грид полностью очищается от содержимого, после чего на него добавляется «Horizontal layout» с компонентом «Datatable» внутри.	Пройдено

Продолжение таблицы 5.1

№	Описание тестового случая	Ожидаемый результат	Полученный результат
25	**Очистка содержимого грида**. 1) перетянуть «Horizontal layout» на грид; 2) перетянуть «Datatable» на созданный «Horizontal layout»; 3) перетянуть «Vertical layout» на грид; 4) перетянуть «Combo» на созданный «Vertical layout»; 5) нажать на троеточие рядом со вкладкой «All» палитры компонентов; 6) в появившемся попапе нажать на «Presets»; 7) левой кнопкой мыши нажать на кнопку «CLEAR LAYOUT»;	Грид полностью очищается от содержимого.	Пройдено

6 РУКОВОДСТВО ПО ИНТЕГРАЦИИ И ИСПОЛЬЗОВАНИЮ

Данное программное средство позиционируется как интегрируемый сервис, а не как самостоятельное приложение. Это значит, что после интеграции интерфейс может измениться согласно предпочтениям разработчиков, интегрирующих данное программное средство в свой продукт. Однако, интерфейс самих компонентов и правила их взаимодействия останутся неизменными.

6.1 Внешний вид приложения

Программное средство создания веб-приложений с помощью готовых графических компонентов делится на три основные части:

– Палитра компонентов (далее component pallet);

– Грид разметка (далее грид или grid);

– Инспектор свойств (далее property inspector).

Опиональные же части могут быть включены в проект разработчиками, интегрирующими данное программное средство в свой продукт.

6.2 Использование палитры компонентов

Внешний вид компонента инспектора свойств показан на рисунке 6.1.

Во вкладке «All» можно найти все предоставляемые в использование виды виджетов:

– простые компоненты (Components);

– лэйауты (Layouts), представляющие собой компоненты-конейтеры для простых компонентов;

– пользовательские шаблоны (Custom templates).

Список компонентов может содержать более одной колонки компонентов, если ему хватает ширины для того, чтобы все было отрисовано.

Каждому компоненту соответсвует иконка, что облегчает навигацию в списке.

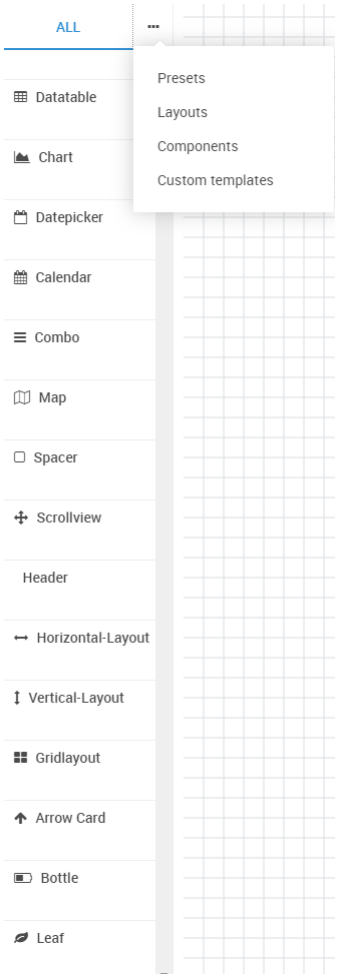

Рисунок 6.1 – Внешний вид палитры компонентов

Выбор определенной вкладки осуществляет фильтрацию данных, предоставляемых списком доступных компонентов, по типу компонента, как показано на рисунке 6.2.

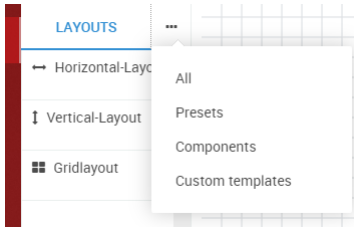

Рисунок 6.2 – Результат фильтрации списка палитры компонентов

6.3 Использование грида

Компонент грид-разметки представлят собой ни что иное, как место, куда пользователь будет перетягивать компоненты, передвигать, настраивать и т.д. Это главный инструмент данного программного средства, по-

зицинирование элементов на нем осуществляется при помощи смещения относительно начала координат: верхней левой точки грида.

Перемещение элементов зависит от размера сетки: чем мельче сетка, тем меньше шаг, с которым могут быть передвинуты компоненты. Ее размер редактируется при помощи панели инструментов, находяшейся сверху:

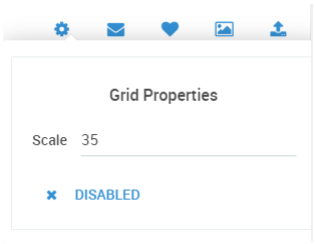

Рисунок 6.3 – Инструмент настройки размеров сетки грида

Также при помощи кнопки «DISABLED/ENABLED» можно выключать/включать отображение сетки.

Сама сетка изначально имеет размер 20 пикселей. Результат изменения размера сетки предсавлен на рисунке 6.4.

Рисунок 6.4 – Результат изменения размеров сетки грида

6.4 Использование инспектора свойств

Компонент инспектор свойств представляет из себя панель инструментов, показанную на рисунке 6.5.

Навигация по вкладкам «DATA», «STYLES» и «CODE» осуществляется путем нажатия на нужную. На каждой странице есть элементы, которые пользователь может сворачивать и разворачивать.

На рисунке 6.5 показано содержимое вкладки «STYLES». Здесь в элеиенте «Formatting» отображаются свойства последнего выделенного компонента.

Кнопки внизу содержимого элемента «Formatting» осуществляют добавление компоненту свойства (кнопка «ADD PROPERTY») и удаление компонента (и всех его дочерних элементов, если такие присутствуют) с грид разметки (кнопка «DELETE COMPONENT»).

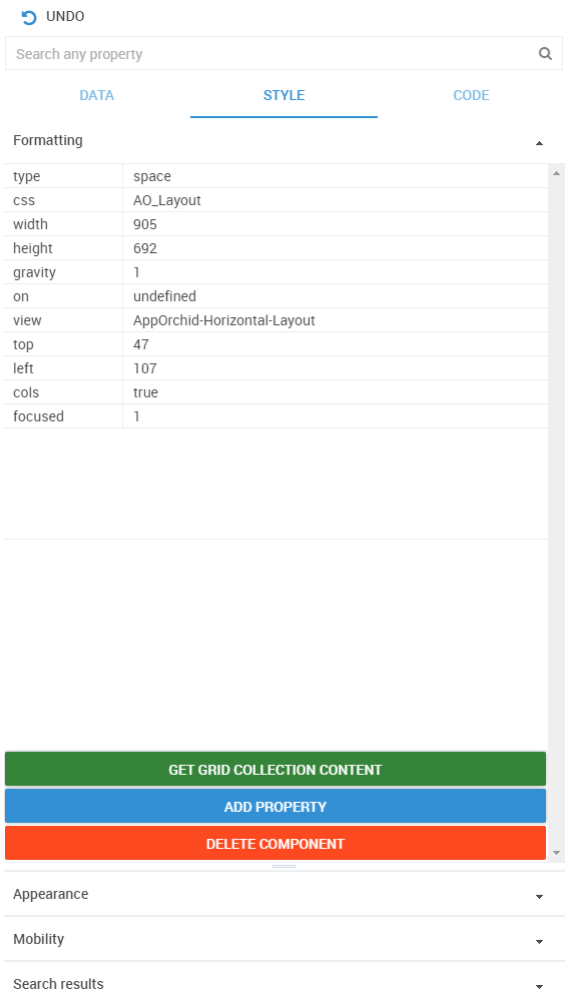

Рисунок 6.5 – Внешний вид инспектора свойств

Вверху компонента присутствует кнопка «UNDO», отменяющая последнее произошедшее действие. Прямо под ней расположена строка поиска свойств. Поиск осуществляется путем ввода нужного значения в поле для ввода и нажатия кнопки Enter. После нажатия будет произведен поиск свойств по ключу и значению, все результаты поиска будут выведены в элементе «Search results», который развернется автоматически.

Результат вышеуказанных действий показан на рисунке 6.6.

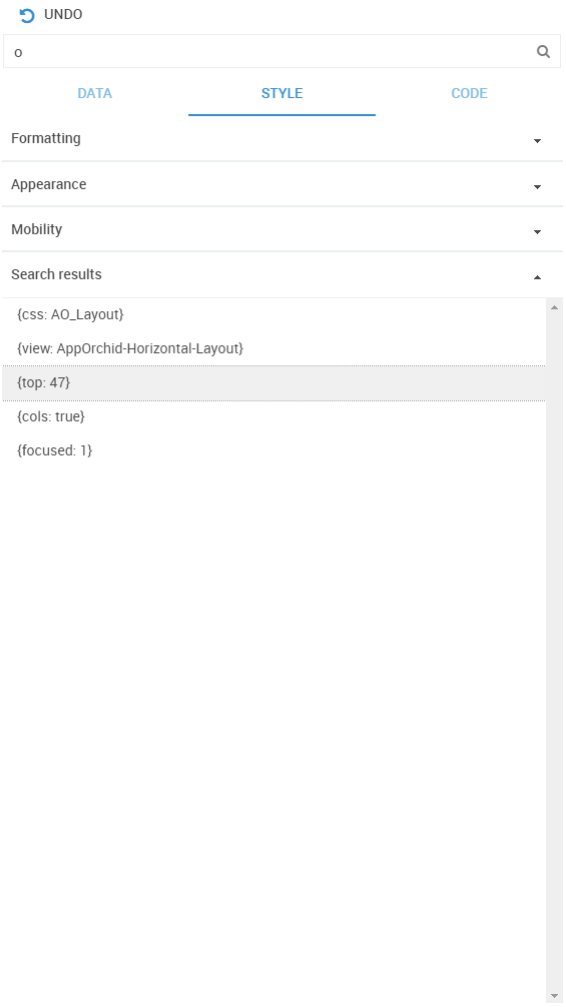

Рисунок 6.6 – Отображение результатов поиска в инспекторе свойств

Результаты поиска предоставляются на основании проведения операции проверки вхождения введенной строки в ключи и значения свойств компонента.

Способ же отображения результатов может быть изменен разработчиками, интегрирующими данное программное средство в свой продукт. Способ отображения по

При нажатии на элемент списка будет открыта соответствующая вкладка со свойствами и выбранное свойство автоматически будет выбрано для редактирования, как показано на рисунке 6.7.

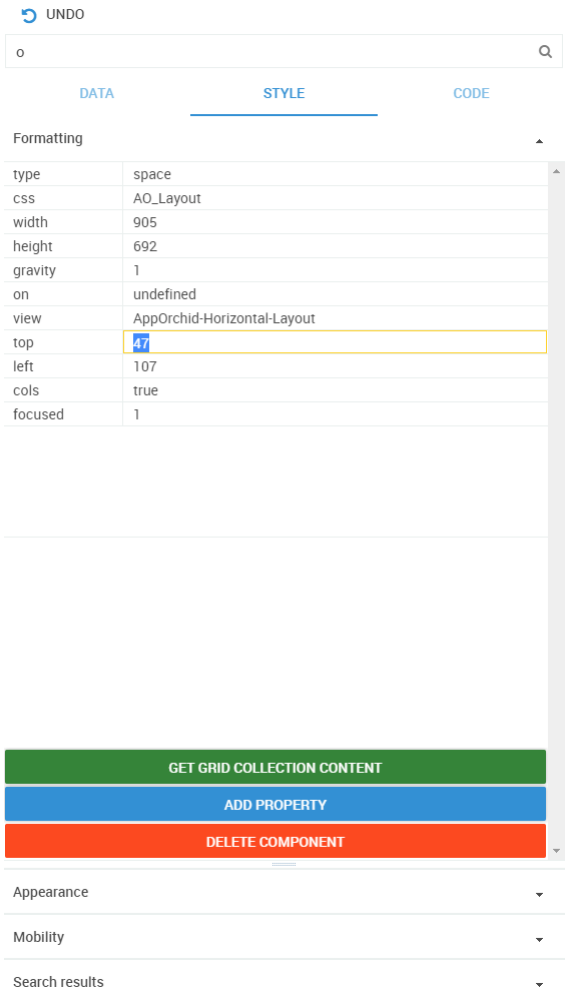

Рисунок 6.7 – Редактирование свойств

Во вкладке «CODE» будет отображаться код объекта конфигурации последнего выделенного компонента. Показанный на рисунке 6.8 код является кодом компонента библиотеки Webix. В данном месте не отображается код дочерних компонентов - это сделано для большей читаемости.

Данные выводятся в формате JSON, как в самом «духовно близком» к языку программирования JavaScript.

Таким образом, можно сделать вывод, что был разработан достаточно гибкий и многофункциональный компонент для работы с «содержимым» компонентов макета, разработанного пользователем приложения. Данный компонент подлежит дальнейшему расширению, так как на данный момент не вся возможная функциональность была реализована в связи с ограничениями во времени разработки.

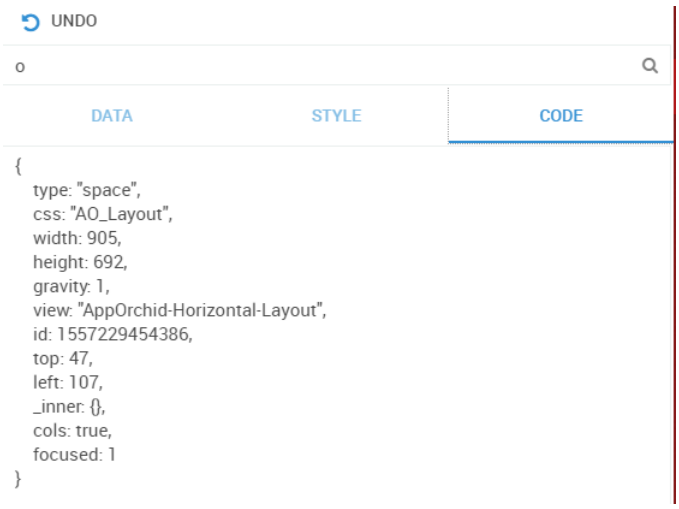

Рисунок 6.8 – Внешний вид вкладки «CODE»

6.5 Использование компонентов и пресетов

Как говорилось ранее, основным инструментом приложения и единственным инструментом для отображения визуальной части создаваемых макетов является грид.

Использование этого алгоритма на практике крайне незатруднительно, что значительно понижает «порог вхождения» пользователей.

6.5.1 Компоненты

Применение компонентов осуществляется по алгоритму «Drag-n-drop», что в переводе с английского означает буквально «Тащи и бросай». Наглядно этот процесс показан на рисунке 6.9.

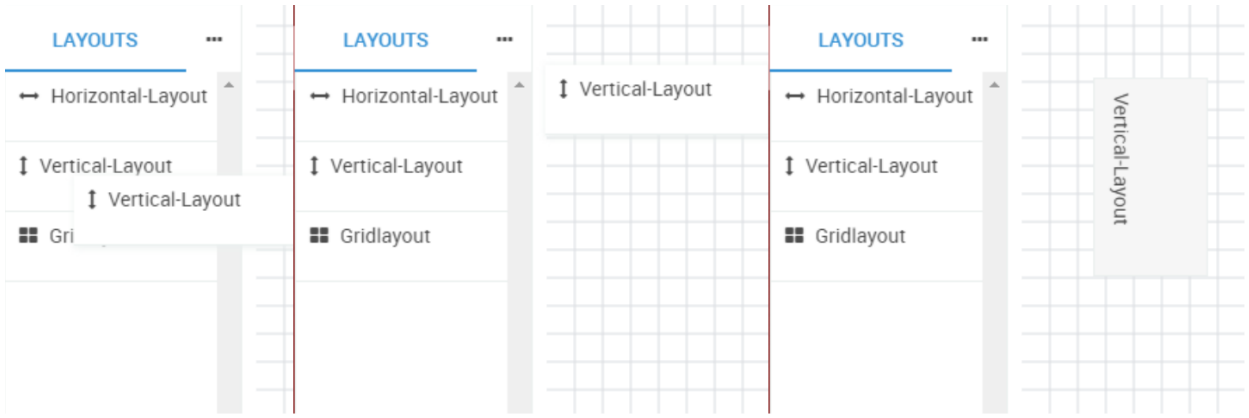

Рисунок 6.9 – Перетягивание компонента на грид

Непосредственно на грид могут быть перетянуты лишь компоненты типа «layout» или пользовательские шаблоны. При попытке перетянуть другие компоненты на грид в правом верхнем углу экрана будет появляться

сообщение об ошибке, как показано на рисунке 6.10.

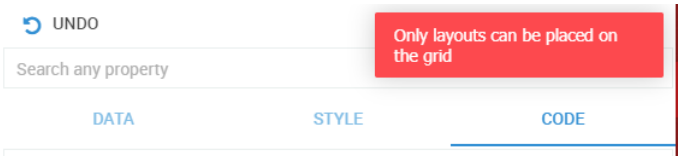

Рисунок 6.10 – Сообщение об ошибке при перетягивании компонента на грид

При перетягивании компонента, не относящегося к типу «layout», на другой компонент, относящийся к типу «layout», последний будет подсвечен (если не будет иметь дочерние элементы в момент перетягивания), как показано на рисунке 6.11.

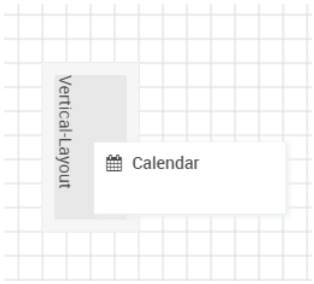

Рисунок 6.11 – Сообщение об ошибке при перетягивании компонента на грид

Результатом процесса перетягивания будет помещение перетягиваемого компонента внутрь объекта, на который он был перетянут. Данный результат продемонстрирован на рисунке 6.11.

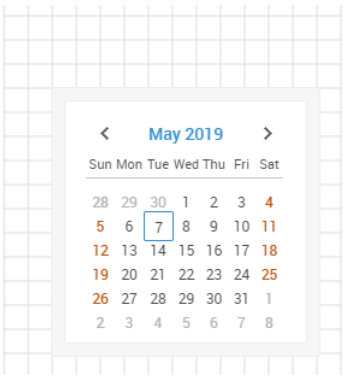

Рисунок 6.12 – Результат перетягивания компонента на лэйаут

Стоит отметить, что данное программное средство не ставит никакие ограничения по вложенности компонентов. То есть, никто и ничто не помешает пользователю помесить лайэаут в лэйаут, который находится в другом лэйауте и т.д. Пример представлен на рисунке 6.13.

Таким образом, можно сделать вывод, что данная функциональность была реализована должным образом и ни в коей мере не будет «сковывать» пользователя в подобного рода действиях.

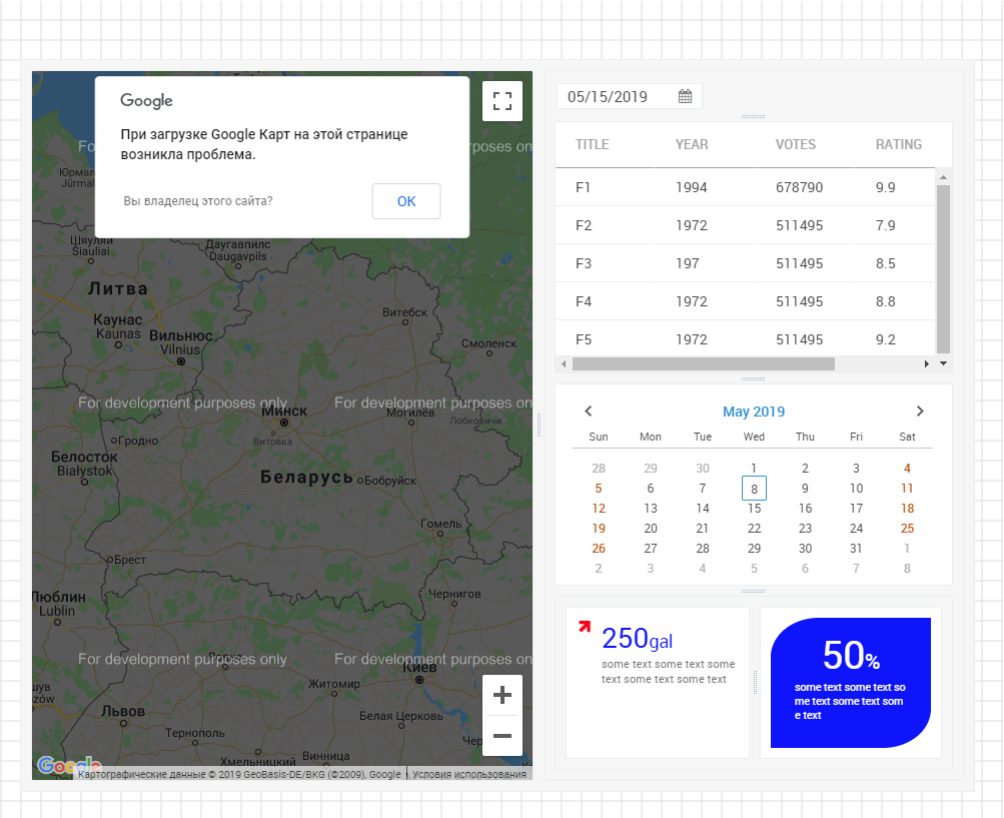

Рисунок 6.13 – Комплексный компонент, состоящий из двух лэйаутов и их содержимого

6.5.2 Пресеты

Как уже говорилось, для использования пресетов существует одноименная вкладка в component pallet. После ее открытия появляется список доступных пресетов, однако если их нет, отобразится соответствующее сообщение, как показано на рисунке 6.14.

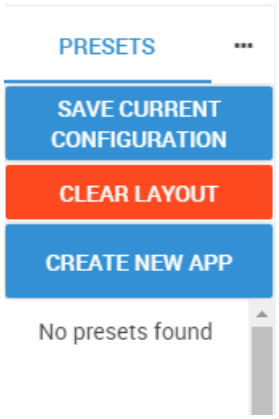

Рисунок 6.14 – Результат перетягивания компонента на лэйаут

Для сохранения текущего макета в пресет необходимо нажать на кнопку «SAVE CURRENT CONFIGURATION», выбрав затем имя для пресета. На данный момент присутствует простейшая валидация на ввод пустой строки, как продемонстрировано на рисунке 6.15.

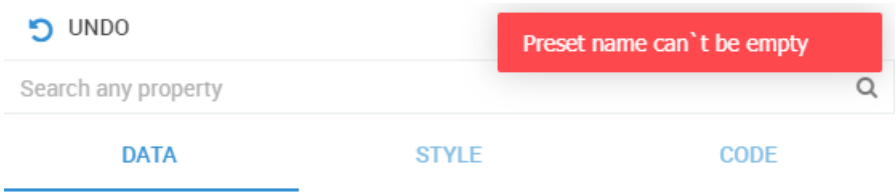

Рисунок 6.15 – Результат перетягивания компонента на лэйаут

В случае, если уже существует пресет с введенным именем, также будет выведено соответствующее сообщение об ошибке, как продемонстрировано на рисунке 6.16.

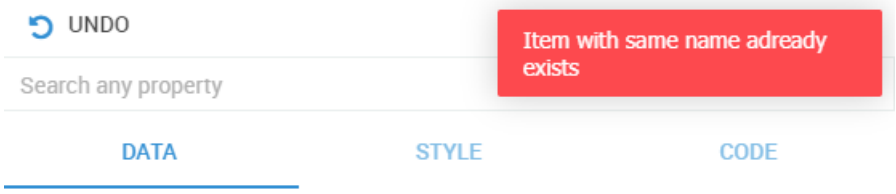

Рисунок 6.16 – Результат перетягивания компонента на лэйаут

Если же введенное имя проходит валидацию, то пресет сохраняется в списке, который сраху же перерисовывается, добавляя новый элемент в самом низу. Содержимое грида при этом не очищается, поэтому пользователь сможет продолжить создание макета.

Результатом применения пресета будет содержимое грида на момент сохранения пресета и соответствующее сообщение в правом верхнем углу экрана, как показано на рисунке 6.17.

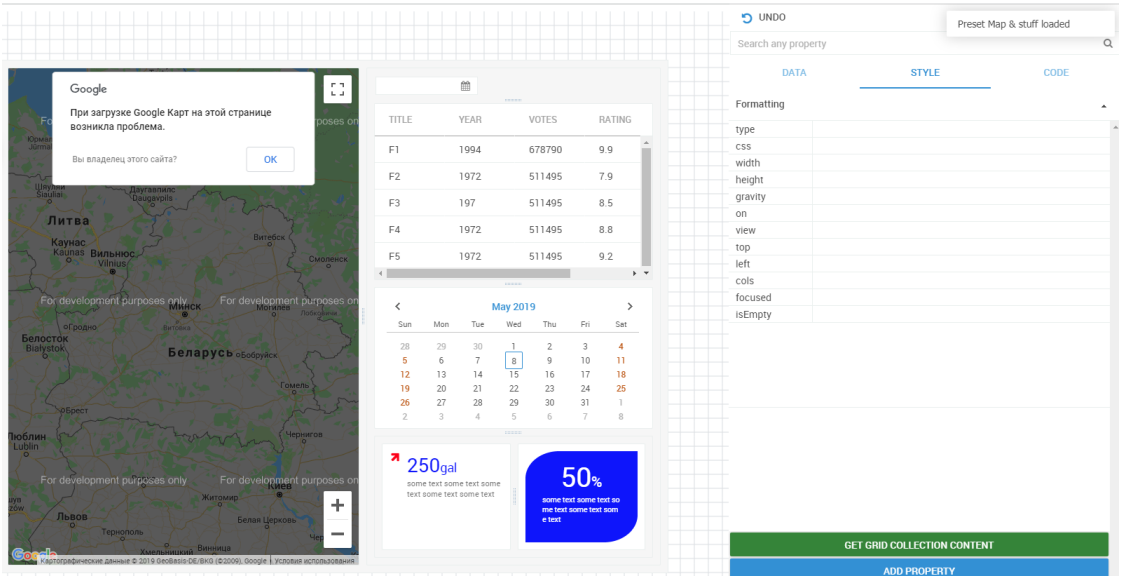

Рисунок 6.17 – Результат применения пресета

6.6 Интеграция программного средства

Программное средство создания веб-приложений с помощью готовых графических компонентов позиционируется как интегрируемый сервис. В конечном счете оно представляет собой набор всего лишь трех основных компонентов: палитры компонентов, грида и инспектора свойств.

Несмотря на всю гибкость предоставляемых возможностей, есть определенные требования к тому, как данное программное средство будет интегрироваться:

– при инициализации можно создать сколько угодно палитр компонентов и инспекторов свойств, но каждый из них может быть связан только с одним гридом;

– при инициализации грид компоненту необходимо указать уникальный идентификатор, а в палитре компонентов и инспекторе свойств указать этот индентификатор в качестве значения поля с ключом «gridId»;

– при инициализации палитры компонентов, необходимо указать свойство с ключом «widgets», значению которого будет соответствовать адрес узла сервера, который предоставит все необходимые данные в нужном формате;

– загружаемые данные должны соответствовать строго определенному формату;

– необходимо указать способ загрузки пресетов и компонентов: можно предоставить непосредственно набор данных или загружать их при помощи самих компонентов.

Спецификация формата данных выгляит следующим образом:

```
{
  config: {
    // config content
  },
  element: 'webix_view_name',
  icon: 'area-chart'
}
```

Краткое текстовое пояснение к спецификации определения формата данных программного средства создания веб-приложений, в частности палитры компонентов, звучит следующим образом:

– поле «config» должно содержать JavaScript-объект - конфигурацию компонента в соответствии с документацией библиотеки Webix;

– значению поля «element» должно соответствовать строковое значение, являющееся названием компонента библиотеки Webix;

– значению поля «icon» должно соответствовать строковое значение, являющееся названием иконки в библиотеке font-awesome.

Что же до пресетов, то, ввиду того, что являются отдельной категорией данных, определение способа их загрузки остается за разработчиком, интегрирующим данное программное средство в свой продукт.

Конфиг для инициализации программного средства в другой продукт в коненом счете будет иметь вид, указанный в участе кода ниже.

```
webix.ui({
    container: 'root',
    cols: [
      {
        view: 'AppOrchid-Component-Pallet',
        widgets: url,
        gridId: 'Grid1',
        width: 170
      },
      {
        id: 'Grid1',
        gravity: 2,
        view: 'AppOrchid-Components-Layout'
      },
      {
        view: 'AppOrchid-Components-Settings',
        gridId: 'Grid1',
        gravity: 1
      }
    ]
});
```

Вместо константы url, показанной выше, следует поместить ссылку на адрес узла сервера, который предоставит все необходимые данные в нужном формате.

Свойство «gravity» нужно для задания пропорций, с которыми будут отображаться компоненты - например, в вышеуказанном коде палитра компонентов имеет фиксированную ширину 170 пикселей, а остальные два компонента делят оставшуюся ширину контейнера в соотношении 2 к 1.

В проекте используются нововведения и синтаксис ES6 [23], ввиду чего в проекте, в который будет интегрироваться данное программное средство, обязан использоваться Babel [7], иначе проект просто не будет работать.

Ниже будет представлен полный список использованных при разработке проекта зависимостей.

```
{
    "dependencies": {
        "body-parser": "^1.18.2",
        "copy-webpack-plugin": "^4.5.1",
        "css-loader": "^0.28.11",
        "express": "^4.16.3",
        "express-fileupload": "^0.4.0",
        "file-loader": "^1.1.11",
        "html-webpack-plugin": "^3.1.0",
        "less": "^3.8.1",
        "less-loader": "^4.1.0",
        "style-loader": "^0.20.3",
        "ts-loader": "^4.5.0",
        "tslint": "^5.11.0",
        "typescript": "^3.0.1",
        "url-loader": "^1.0.1",
        "webpack-dev-server": "^3.1.5"
    },
    "devDependencies": {
        "babel-core": "^6.26.0",
        "babel-loader": "^7.1.4",
        "babel-plugin-module-resolver": "^3.1.1",
        "babel-plugin-transform-object-rest-spread": "^6.26.0",
        "babel-polyfill": "^6.26.0",
        "babel-preset-env": "^1.6.1",
        "eslint": "^4.19.1",
        "eslint-plugin-import": "^2.9.0",
        "eslint-xbsoftware": "git+https://bitbucket.org/xbsltd/
          eslint.git",
        "extract-text-webpack-plugin": "^4.0.0-beta.0",
```

```
        "resolve-url-loader": "^2.3.0",
        "webpack": "^4.16.5",
        "webpack-cli": "^2.0.13"
    }
}
```

Резюмируя всё вышесказанное, можно сделать вывод, что при разработке приложения применялись приемы, благоприятно воздействующие на UX, а также обеспечивающие более легкую разработку, возможное будущее расширение и поддержку приложения.

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ВНЕДРЕНИЯ ПРОГРАММНОГО СРЕДСТВА

7.1 Характеристика программного средства

Разрабатываемый программное средство создания веб-приложений с помощью готовых графических компонентов представляет собой интегрируемый программный модуль. Разрабатываемый модуль позволяет:

– чистая дисконтированная стоимость (ЧДД);
– срок окупаемости инвестиций (ТОК);
– рентабельность инвестиций (Ри).

Разработка программного модуля осуществлялась на языке JavaScript с использованием технологии Webix.

Разработанное программное средство предполагает, что его интеграция будет производиться в приложение, построенное на основе архитектуры клиент – сервер, где сервером является web-сервер, а клиентом приложение, в которое будет интегрироваться данный программное средство. При инициализации программный модуль будет отправлять запрос на сервер с целью получения конфигураций базовых компонентов, которые и будут использоваться конечным пользователем при создании своего макета приложения.

Интегрируемый программный модуль позволяет разработчику приложения, являющегося местом интеграции, дать пользователю возможность самому определить внешний вид приложения согласно своим предпочтениям. Набор базовых компонентов и правила создания макета определяет разработчик, в чей продукт будет интегрироваться данное программное средство.

Разрабатываемый программный модуль имеет следующие преимущества:

– простой и понятный интерфейс;
– легко конфигурируется;
– гибкая интеграция.

7.2 Расчет затрат на разработку и отпускной цены программного модуля

Общий объем программного модуля определяется на основе информации о функциях разрабатываемого программного модуля, исходя из количества и объема функций, реализуемых программным модулем, по формуле:

$$\mathrm{V_o} = \sum_{i=1}^{n} \mathrm{V}_i, \tag{7.1}$$

где V_i — объем i-ой функции программного модуля (количество строк исходного кода (LOC));
n — общее число функций.

С учетом условий разработки общий объем программного модуля уточняется в организации и определяется уточненный объем программного модуля по формуле:

$$\mathrm{V}_y = \sum_{i=1}^{n} \mathrm{V_{yi}}, \tag{7.2}$$

где $\mathrm{V_{yi}}$ — уточненный объем i-й функции программного модуля (LOC).

Расчет объема функций программного средства и общего объема приведен в таблице 7.1.

Таблица 7.1 – Перечень и объём функций программного модуля

№ функции	Наименование (содержание)	Объём функции, LoC
101	Организация ввода информации	300
102	Контроль, предварительная обработка и ввод информации	500
109	Организация ввода/вывода информации в интерактивном режиме	190
111	Управление вводом/выводом	600
207	Манипулирование данными	4000
506	Обработка ошибочных и сбойных ситуаций	500
507	Обеспечение интерфейса между компонентами	750
601	Отладка прикладных программ в интерактивном режиме	1300
707	Графический вывод результатов	4500
	Общий объем	11 640

В связи с использованием более совершенных средств автоматизации объемы функций были уменьшены и уточненный объем программного модуля составил 10500 LOC вместо 11640 LOC.

Программный модуль относится ко второй категории сложности, и, следовательно, нормативная трудоемкость составит 78 чел./дн.

Коэффициенты использования стандартных модулей и новизны программного модуля, составят $К_т = 0{,}06$ и $К_н = 0{,}07$ соответственно.

Коэффициент сложности программного модуля $К_с$ рассчитывается по формуле:

$$К_с = 1 + \sum_{i=1}^{n} К_i = 1 + 0{,}06 + 0{,}07 = 1{,}13, \qquad (7.3)$$

где $К_i$ — нормативная трудоемкость разработки программного модуля (78 чел./дн.);

n — количество учитываемых характеристик.

Нормативная трудоемкость служит основой для определения общей трудоемкости разработки программного модуля, который определяется по формуле:

$$Т_о = Т_н \cdot К_с \cdot К_т \cdot К_н = 78 \cdot 0{,}6 \cdot 0{,}8 \cdot 1{,}13 = 49 \text{ чел./дн.}, \qquad (7.4)$$

где $Т_н$ — коэффициент, соответствующий степени повышения сложности за счет конкретной характеристики;

$К_т$ — поправочный коэффициент, учитывающий степень использования при разработке стандартных модулей;

$К_н$ — коэффициент, учитывающий степень новизны программного модуля;

$К_с$ — коэффициент, учитывающий сложность программного модуля.

В соответствии с договором срок разработки – 1,5 месяца (0,16 г.).

Эффективный фонд времени работы одного человека рассчитывается по формуле:

$$Ф_{эф} = Д_г - Д_п - Д_в - Д_о = 365 - 9 - 103 - 24 = 229 \text{ дн.}, \qquad (7.5)$$

где $Д_г$ — количество дней в году;

$Д_п$ — количество праздничных дней в году;

$Д_в$ — количество выходных дней в году;

$Д_о$ — количество дней отпуска.

Численность исполнителей проекта($Ч_р$) рассчитывается по формуле:

$$Ч_р = \frac{Т_о}{Т_р \cdot Ф_{эф}} = \frac{49}{0{,}16 \cdot 229} = 2 \text{ чел.}, \qquad (7.6)$$

где $Т_о$ — общая трудоемкость разработки проекта, чел./дн.;
$Т_р$ — срок разработки проекта, лет;
$Ф_{эф}$ — эффективный фонд времени работы одного человека.

Разработкой программного средства создания веб-приложений с помощью готовых графических компонентов занимаются двое исполнителей: фронт-энд разработчик и тестировщик.

7.3 Расчет сметы затрат

Расчет основной заработной платы исполнителей представлен в таблице 7.2

Таблица 7.2 – Работники, занятые в проекте

Исполнители	Эффективный фонд времени работы, дней	Дневная тарифная ставка, руб.	Месячный оклад, руб.
Фронт-энд разработчик	35	90	3150
Тестировщик	20	60	1200
Всего	55		4350
Премия (30%)			1305
Основная заработная плата			5655

Дополнительная заработная плата включает выплаты, предусмотренные законодательство о труде: оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей, и определяется по нормативу, установленному в организации, в процентах к основной заработной плате [26]. Приняв данный норматив $Н_д = 20\%$, рассчитаем дополнительные выплаты:

$$З_д = \frac{З_о \cdot Н_д}{100\%} = \frac{5655 \cdot 10\%}{100\%} = 565{,}5 \text{ руб.}, \tag{7.7}$$

где $З_о$ — норматив дополнительной заработной платы (10%).

Отчисления в фонд социальной защиты населения и в фонд обязательного страхования определяются в соответствии с действующим законодательством по нормативу в процентном отношении к фонду основной и дополнительной зарплат по следующим формулам:

$$З_{сз} = \frac{(З_о + З_д) \cdot Н_{сз}}{100\%} = \frac{(5655 + 565{,}5) \cdot 34{,}6\%}{100\%} = 2152{,}3 \text{ руб.}, \tag{7.8}$$

где $З_о$ — норматив дополнительной заработной платы (10%);
$З_д$ — дополнительная заработная плата, руб.;
$Н_{сз}$ — норматив отчислений в фонд социальной защиты населения и на обязательное страхование (34 + 0,6%).

Расходы по статье «Материалы» рассчитываются по формуле:

$$М = Н_м \cdot \frac{V_о}{100} = 0{,}5 \cdot \frac{10500}{100} = 52{,}5 \text{ руб.}, \quad (7.9)$$

где $Н_м$ — норма расхода материалов в расчете на 100 строк исходного кода программного модуля (0,5 руб.);
$V_о$ — общий объем программного модуля (10500 LoC).

Расходы по статье «Машинное время» включают оплату машинного времени, необходимого для разработки и отладки ПО, которое определяется по нормативам на 100 строк исходного кода и рассчитываются по формуле:

$$Р_м = Ц_м \cdot \frac{V_о}{100} \cdot Н_{мв} = 0{,}3 \cdot \frac{11\,640}{100} \cdot 12 = 419 \text{ руб.}, \quad (7.10)$$

где $Ц_м$ — цена одного машино-часа (0,3 руб.);
$Н_{мв}$ — норматив расхода машинного времени на отладку 100 строк исходного кода (12 машино-часов).

Затраты по статье «Накладные расходы», связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды, относятся к конкретному ПО по нормативу в процентном отношении к основной заработной плате исполнителей. Принимая норматив равным $Н_{нр} = 80\%$ получим величину расходов:

$$Р_н = \frac{З_о \cdot Н_{нр}}{100\%} = \frac{5655 \cdot 80\%}{100\%} = 4524 \text{ руб.} \quad (7.11)$$

Общая сумма расходов по смете определяется как сумма вышерассчитанных показателей и рассчитывается по формуле:

$$С_п = З_о + З_д + З_{сз} + З_{ос} + М + Р_м + Р_{нк} + П_з + Р_н = \\ = 13\,368{,}3 \text{ руб.} \quad (7.12)$$

7.4 Расчет экономической эффективности реализации на рынке программного средства создания веб-приложений

Программный модуль планируется разместить в сети Интернет, организации, имеющие в структуре курьерскую службу доставки грузов, смогут скачать его по цене, определенной на основе маркетинговых исследований рынка систем управления контентом сайта, 2400 руб.

В среднем в год прогнозируется, что 15 организаций скачает разработанный программный модуль.

Налог на добавленную стоимость составит:

$$\text{НДС} = \frac{36000 \cdot 20\%}{100\%} = 7200 \text{ руб.}. \tag{7.13}$$

Таким образом, разработчик получит доход в размере 36000 руб. Прирост чистой прибыли от реализации программного модуля в сети Интернет можно определить по формуле 7.14. В первый год реализации программного модуля на рынке разработчик получит чистую прибыль в размере::

$$\begin{gathered}\Delta\text{П} = (\text{Д} - \text{НДС} - \text{З}) \cdot (1 - \text{Н}_\text{п}) = \\ = (36000 - 7200 - 13\,368{,}3) \cdot (1 - 0{,}18) = 11\,110{,}8 \text{ руб.},\end{gathered} \tag{7.14}$$

где Д — доход полученный от реализации продукта, руб.;
НДС — налог на добавленную стоимость (20%);
З — затраты на разработку программного модуля, руб.;
$\text{Н}_\text{п}$ — ставка налога на прибыль (18%).

Рентабельность затрат на разработку программного средства создания веб-приложений с помощью готовых графических компонентов составит:

$$\text{Р} = \frac{\Delta\text{П}}{\text{С}_\text{п}} \cdot 100\% = \frac{11\,110{,}8}{13\,368{,}3} \cdot 100\% = 83{,}1\%, \tag{7.15}$$

Таким образом, все затраты на разработку программного модуля окупятся в первый год от реализации программного модуля в сети Интернет. Преимущества разработанного модуля для пользователя позволяют прогнозировать его коммерческий успех в будущем.

Следовательно, реализация программного модуля автоматизации курьерской службой на рынке является экономически эффективной и его разработку целесообразно осуществлять.

ЗАКЛЮЧЕНИЕ

В ходе разработки были исследованы особенности программных средств категории конструкторов сайтов, парадигмы разработки на языке программных средств на языке программирования JavaScript, в частности функциональная парадигма.

Был произведен анализ предметной области, выбраны основные технологии и инструменты для разработки, разработаны функциональные требования к программному средству. Позднее на их основании было произведено проектирование разрабатываемого программного средства.

В процессе проектирования был принят ряд решений об использовании определенных паттернов проектирования [27], был разработан основной алгоритм работы программного средства и определена его архитектура, в качестве которой выступает SPA [14].

С целью проверки соответствия функциональным требованиям и корректности работы программного средства в целом была разработана серия тестовых сценариев, которые впоследствии были успешно пройдены: все получаемые результаты работы программного средства полностью совпадали с ожидаемыми.

Результаты тестирования свидетельствуют о том, что разработанное программное средство полностью соответствует требованиям и будет функционировать корректно, согласно предъявляемым к нему требованиям.

Помимо вышеперечисленного, разработке данного программного средства было дано технико-экономическое обоснование, включающее в себя расчет экономической эффективности разработки и реализации программного средства создания веб-приложений. Целесообразность разработки была подтверждена, а расчеты показали, что вложенные в проект финансовые инвестиции, окупятся за 14 месяцев.

По итогам разработки и тестирования можно сделать вывод, что разработанное программное средство разработано и функционирует в соответствии с предъявляемыми требованиями. Важным является тот факт, что приложение легко расширяется в контексте функциональности, а также предоствляет разработчикам возможность самим определять необходимую надстраиваемую поверх базовой функциональность.

Подытожив, можно сделать вывод, что разработанное программное средство соответствует задачам, поставленным в дипломном проектировании. Отсюда следует, что программное средство может быть поставлено на рынок. Цели дипломного проекта можно считать достигнутыми.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Апплет [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ru.wikipedia.org/wiki/Апплет — Дата доступа: 27.04.2019.

[2] Полное руководство по CSS Grid. — Электронные данные. — Режим доступа: https://tuhub.ru/posts/css-grid-complete-guide — Дата доступа: 01.05.19.

[3] Callback (программирование) [Электронный ресурс]. — Электронные данные. — Режим доступа: https://en.wikipedia.org/wiki/Callback_(computer_programming)— Дата доступа: 28.04.2019.

[4] Полное руководство по CSS Grid. — Электронные данные. — Режим доступа: https://ru.wikipedia.org/wiki/Лейаут — Дата доступа: 01.05.19.

[5] Полное руководство по CSS Grid. — Электронные данные. — Режим доступа: https://en.wikipedia.org/wiki/Serialization — Дата доступа: 01.05.19.

[6] AJAX [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ru.wikipedia.org/wiki/AJAX — Дата доступа: 28.04.2019.

[7] Babel [Электронный ресурс]. — Электронные данные. — Режим доступа: https://babeljs.io — Дата доступа: 27.04.2019.

[8] Content Management System [Электронный ресурс]. — Электронные данные. — Режим доступа: https://otvet.mail.ru/question/16298036 — Дата доступа: 28.04.2019.

[9] Document Object Model [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ru.wikipedia.org/wiki/Document_Object_Model — Дата доступа: 28.04.2019.

[10] Drag-and-drop [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ru.wikipedia.org/wiki/Drag-and-drop — Дата доступа: 28.04.2019.

[11] Git и GitHub: что это такое [Электронный ресурс]. — Электронные данные. — Режим доступа: https://tproger.ru/translations/difference-between-git-and-github — Дата доступа: 27.04.2019.

[12] Интерфейс пользователя [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ru.wikipedia.org/wiki/Интерфейс_пользователя — Дата доступа: 28.04.2019.

[13] REST [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ru.wikipedia.org/wiki/REST — Дата доступа: 09.05.2019.

[14] Одностраничное приложение [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ru.wikipedia.org/wiki/Одностраничное_приложение — Дата доступа: 09.05.2019.

[15] Условно-бесплатное программное обеспечение [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ru.wikipedia.org/wiki/Условно-бесплатное_программное_обеспечение — Дата доступа: 09.05.2019.

[16] Что такое UX/UI дизайн на самом деле? [Электронный ресурс]. — Электронные данные. — Режим доступа: https://habr.com/ru/post/321312/ — Дата доступа: 09.05.2019.

[17] K., Simpson. You Don't Know JS / Simpson K. — O 'Reilly Media, 2015.

[18] K., Simpson. Functional-Light JavaScript / Simpson K. — frontendmasters.com, 2017.

[19] Webix Documentation [Электронный ресурс]. — Электронные данные. — Режим доступа: https://docs.webix.com — Дата доступа: 09.05.2019.

[20] Конструкторы сайтов: обзор 8 лучших сервисов со сравнительной таблицей [Электронный ресурс]. — Электронные данные. — Режим доступа: https://texterra.ru/blog/konstruktory-saytov-obzor-8-luchshikh-servisov-so-sravnitelnoy-tablitsey.html — Дата доступа: 27.04.2019.

[21] JavaScript [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ru.wikipedia.org/wiki/JavaScript — Дата доступа: 27.04.2019.

[22] ECMAScript [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ru.wikipedia.org/wiki/ECMAScript — Дата доступа: 27.04.2019.

[23] ES6, ES8, ES2017: что такое ECMAScript и чем это отличается от JavaScript [Электронный ресурс]. — Электронные данные. — Режим доступа: https://tproger.ru/translations/wtf-is-ecmascript/ — Дата доступа: 27.04.2019.

[24] ES6, ES8, ES2017: что такое ECMAScript и чем это отличается от JavaScript [Электронный ресурс]. — Электронные данные. — Режим доступа: https://habr.com/ru/post/256505/ — Дата доступа: 27.04.2019.

[25] Шпаргалка по шаблонам проектирования [Электронный ресурс]. — Электронные данные. — Режим доступа: https://habr.com/ru/post/210288/ — Дата доступа: 27.04.2019.

[26] Палицын, В.А. Технико-экономическое обоснование дипломных проектов: Метод. пособие для студ. всех спец. БГУИР. В 4-х ч. Ч. 4: Про-

екты программного обеспечения / В.А. Палицын. — Мн : БГУИР, 2006. — 76 с.

[27] Шаблон проектирования [Электронный ресурс]. — Электронные данные. — Режим доступа: https://ru.wikipedia.org/wiki/Шаблон_проектирования — Дата доступа: 27.04.2019.

ПРИЛОЖЕНИЕ А
(обязательное)
Фрагменты исходного кода

```
import BaseLayout from '../interfaces/BaseLayout';
import getGrid from '../helpers/getGrid';
import DragContext from '../interfaces/subinterfaces/DragContext';

const baselayoutConfig: BaseLayout = {
   name: 'AppOrchidBaseLayout',
   defaults: {
      isEmpty: true,
      type: 'space',
      css: 'AO_Layout'
   },
   $init() {
      this.$ready.push(
         () => {
            this.attachEvent('_onElementRemoved', function () {
               let flag = 0;
               let children: webix.ui.layout[] = this.getChildViews();

               if (!children.length) {
                  this.getParentView().getParentView().removeView(this.
                     getParentView())
                  return;
               }

               if (children[0].config.view === 'resizer') {
                  this.blockEvent();
                  this.removeView(children[0].config.id);
                  this.unblockEvent();
               }

               if (children[children.length - 1].config.view === 'resizer') {
                  this.blockEvent();
                  this.removeView(children[children.length - 1].config.id);
                  this.unblockEvent();
               }
               children.forEach((element) => {
                  if (flag && element.config.view === 'resizer') {
                     const elementParent = element.getParentView();

                     elementParent.blockEvent();
                     elementParent.removeView(element.config.id);
                     elementParent.unblockEvent();
                     flag = 0;
                  } else if (element.config.view === 'resizer') {
                     flag = 1;
                  }
               });

            });
         }
```

```
    );
},
$drop(source, target, ev) {
    let dnd: DragContext = webix.DragControl.getContext();
    if (dnd.from.config) {
        const grid = <webix.ui.gridlayout>getGrid(this);
        const webixTarget = <webix.ui.layout>this._getWebixElementFromNode(ev
            .target);

        let elementId: string | number;
        let elementConfig: any = webix.copy(dnd.from.getItem(dnd.source[0]).
            config);
        elementConfig.parentId = this.config.id;
        elementConfig.id = webix.uid();

        if (this.config.isEmpty || webixTarget === this) {
            elementId = this._addElement(elementConfig);
        } else {
            let position = -1;
            for (let i = 0; i < this.getChildViews().length; i++) {
                if (this.getChildViews()[i] === webixTarget) {
                    position = i;
                    break;
                }
            }

            elementId = this._addElement(elementConfig, position);
        }
        grid.callEvent('componentAdded', [elementId]);
    }
},
_getWebixElementFromNode(node) {
    if (node.getAttribute('view_id')) {
        return <webix.ui.layout>$$(node.getAttribute('view_id'));
    }

    return <HTMLElement>this._getWebixElementFromNode(node.parentNode);
},
_addElement(config, position) {
    this.callEvent('newElementAdded');
    config.on = {
        onFocus(view) {
            const grid = <webix.ui.gridlayout>getGrid(view);
            grid.callEvent('onMovableElementSelect', [view]);
        }
    };

    if (this.config.isEmpty) {
        delete this.config.isEmpty;
        return webix.ui(webix.copy(config), this.getChildViews()[0]);
    }
    if (this.getChildViews().length > 0) {
        this.addView({
            view: 'resizer'
        }, position);
```

```
        }

        return this.addView(webix.copy(config), position);
    }
};

export default baselayoutConfig;

import baselayoutConfig from './config';

webix.protoUI(baselayoutConfig, webix.ui.layout);

import { callWindowOnGridlayoutDrop } from './helpers/gridLayoutWindow/config'
    ;
import ComponentsLayout from '../layouts/interfaces/ComponentsLayout';
import { deleteSelectedCss } from '../../helpers/highlighting';
import { deleteResizer } from './helpers/componentResizer';


// const UNFOCUSED = 0;
const FOCUSED = 1;
const DEFAULT_OFFSET = 400;


function getBranch(origin, parentId) {
    return origin.filter(element => element.parentId === parentId);
}

const componentLayout: ComponentsLayout = {
    name: 'AppOrchid-Components-Layout',
    default: {
        drag: 'target',
        cells: []
    },
    config: {
        separation: 20
    },
    _canvasId: `canvas:grid:${webix.uid()}`,
    $init(config) {
        const self = this;
        // ignore serialize for itself
        config.presetParse = false;

        this._collection = [];
        this._cells = [];

        webix.DragControl.addDrop(self.$view, {
            $drop(source, target, ev) {
                let dnd = webix.DragControl.getContext();

                if (dnd.from.name === 'dataview' &&
                    (dnd.from.getItem(dnd.source[0]).widgetType === 'layout' || dnd
                        .from.getItem(dnd.source[0]).widgetType === 'card')
                ) {
                    if (dnd.from.name === 'dataview') {
```

```
                let elementConfig = webix.copy(dnd.from.getItem(dnd.source
                    [0]).config);

                if (elementConfig.view === 'AppOrchid-Gridlayout') {
                   self._onGridLayoutDrop(elementConfig, ev);
                   return;
                }

                elementConfig.id = webix.uid();
                self._addMovableElement(elementConfig, ev);
             }
          } else if (dnd.from.name === 'dataview' && dnd.from.getItem(dnd.
             source[0]).widgetType !== 'layout') {
             webix.message({
                text: 'Only layouts can be placed on the grid',
                type: 'error',
                expire: 2000
             });
          }
       }
    });
    this.$ready.push(this.drawCanvasGrid);
},
_onGridLayoutDrop(elementConfig, ev) {
    let promise = new Promise((resolve, reject) => {
       callWindowOnGridlayoutDrop(resolve, reject);
    });

    promise
       .then((res: { dx, dy }) => {
          let { dx, dy } = { dx: parseInt(res.dx, 10), dy: parseInt(res.dy,
             10) };
          elementConfig.height = 2 * dy * this.config.separation;
          elementConfig.width = 2 * dx * this.config.separation;

          let gridlayout = Object.assign({}, { gridColumns: dx, gridRows: dy
              }, elementConfig);

          gridlayout.id = webix.uid();
          this._addMovableElement(gridlayout, ev);
       })
       .catch(err => webix.message({
          text: err,
          type: 'error',
          expire: 2000
       }));
},
drawCanvasGrid() {
    const self = this;
    let conf = {
       template: () => `<canvas
                      id ="${self._canvasId}"
                      height="${document.body.clientHeight}"
                      width="${document.body.clientWidth}"
                      />`,
```

```
            top: 0,
            left: 0,
            zIndex: -10,
            autoheight: true,
            presetParse: false,
            on: {
               onAfterRender() {
                  this.define('height', document.body.clientHeight);
                  this.define('width', document.body.clientWidth);
                  this.resize();

                  self._drawGrid();
               }
            }
         };
         this.addView(conf);
      },
      _drawGrid(gridOptions) {
         let cnv = this.getNode()
            .querySelector('canvas');
         // let cnv = document.getElementById(this._canvasId);
         let opts = gridOptions;

         /* if (!opts) {
            opts = {
               lines: {
                  separation: 20,
                  color: '#DFE2E6'
               }
            };
         }
         this.config.separation = opts.lines.separation;
         this._drawGridLines(cnv, opts.lines); */

         if (!opts) {
            opts = {
               separation: 20,
               color: '#DFE2E6'
            };
         }
         this.config.separation = opts.separation;
         this._drawGridLines(cnv, opts);
      },
      _drawGridLines(cnv, lineOptions) {
         let iWidth = cnv.width;
         let iHeight = cnv.height;

         let iCount;
         let ctx: CanvasRenderingContext2D = cnv.getContext('2d');
         ctx.strokeStyle = lineOptions.color;
         /* ctx.strokeWidth = 1; */
         ctx.beginPath();

         iCount = Math.floor(iWidth / lineOptions.separation);
```

```
    for (let i = 1; i <= iCount; i++) {
        let x = i * lineOptions.separation;
        ctx.moveTo(x, 0);
        ctx.lineTo(x, iHeight);
    }

    iCount = Math.floor(iHeight / lineOptions.separation);

    for (let i = 1; i <= iCount; i++) {
        let y = i * lineOptions.separation;
        ctx.moveTo(0, y);
        ctx.lineTo(iWidth, y);
    }

    ctx.strokeStyle = lineOptions.color || '#DFE2E6';
    ctx.fillStyle = 'rgba(209,209,209,0);';

    ctx.stroke();
    ctx.closePath();
},
_getMovableElementEvents() {
    const self = this;
    const movableElementEvents = {
        onFocus(view) {
            this.config.focused = FOCUSED;
            const grid = <webix.ui.baselayout>$$(self.config.id);

            grid.callEvent('onMovableElementSelect', [view, self]);
        },
        onBlur() {
            deleteSelectedCss();
            deleteResizer();
        }
    };

    return {
        on: { ...movableElementEvents }
    };
},
_addMovableElement(config, ev) {
    let height;
    let width;
    let top;
    let left;
    const self = this;

    // when vertical or horizontal layout
    if (!config.width || !config.height) {
        width = this.config.separation;
        height = this.config.separation;
    } else {
        width = config.width;
        height = config.height;
    }
```

```
top = ev ? ev.offsetY - (height / 2) : DEFAULT_OFFSET;
left = ev ? ev.offsetX - (width / 2) : DEFAULT_OFFSET;

config.top = top;
config.left = left;

function _recursiveMovableEvents(element) {
   if (element.cols && element.cols.length) {
      element.cols = element.cols.map(subElement =>
         _recursiveMovableEvents(subElement));
   }

   if (element.rows && element.rows.length) {
      element.rows = element.rows.map(subElement =>
         _recursiveMovableEvents(subElement));
   }

   if (element.cells && element.cells.length) {
      element.cells = element.cells.map(subElement =>
         _recursiveMovableEvents(subElement));
   }

   // TODO: add fix for scroll view

   // if (element.cells && element.cells.length) {
   // element.cells = element.cells.map((subElement) => {
   // return _recursiveMovableEvents(subElement);
   // });
   // }

   return Object.assign(
      {},
      self._getMovableElementEvents(element),
      webix.copy(element)
   );
}

let baseElement = {
   view: 'elementMovable',
   move: true,
   presetParse: false,
   css: {
      border: '3px solid transparent'
   },
   rows: [
      _recursiveMovableEvents(config)
   ],
   id: webix.uid(),
   top,
   left
};

this.addView(baseElement);
this.callEvent('componentAdded', [baseElement.id]);
return config;
```

```
},
getSerializedCollection() {
   let allItems = [];

   function __addSubItemsToCollection(parentElement) {
      let childViews = parentElement.getChildViews();
      let parentId = null;
      if (parentElement.hasOwnProperty('config')
         && (!parentElement.config.hasOwnProperty('presetParse')
            || parentElement.config.presetParse === true)) {
         parentId = parentElement.config.id;
      }

      childViews.forEach((singleView) => {
         __addSubItemsToCollection(singleView);

         if (singleView.hasOwnProperty('config')
            && (!singleView.config.hasOwnProperty('presetParse')
               || singleView.config.presetParse === true)) {
            let newConfig = { ...singleView.config };

            if (newConfig.view === 'datatable') {
               delete newConfig.columns;

               newConfig.columns = singleView.config.columns.map((column,
                   index) => ({
                  header: column.header[0].text,
                  id: column.id,
                  minWidth: column.minWidth
               }));
            }

            delete newConfig.suggest;
            delete newConfig.popup;
            if (newConfig.options) {
               newConfig.options = (<any>$$(newConfig.options).config).body
                   .data;
            }

            if (parentId) {
               newConfig.parentId = parentId;
            }
            allItems.push(newConfig);
         }
      });
   }

   __addSubItemsToCollection(this);

   allItems = allItems.reverse();
   let collection = allItems;
   let array = [];
   collection.forEach((curr) => {
      if (curr.view === 'AppOrchid-Horizontal-Layout') {
         curr.cols = getBranch(collection, curr.id);
```

```
            curr.cols = curr.cols.reverse();
            if (curr.cols.length) {
               curr.isEmpty = false;
            }
         } else if (curr.view === 'AppOrchid-Vertical-Layout' || 'AppOrchid-
            Scrollview') {
            curr.rows = getBranch(collection, curr.id);
            curr.rows = curr.rows.reverse();
            if (curr.rows.length) {
               curr.isEmpty = false;
            }
         } else if (curr.view === 'AppOrchid-Gridlayout') {
            curr.cells = getBranch(collection, curr.id);
            curr.cells = curr.cells.reverse();
            if (curr.cells.length) {
               curr.isEmpty = false;
            }
         }
         if (!curr.parentId) {
            array.push(curr);
         }
      });

      return array.reverse();
   },
   addViewMovable(config, ev) {
      return this._addMovableElement(config, ev);
   }
};

export default componentLayout;
```

```
import './helpers/movable';
import componentLayout from './config';
import './helpers/gridLayoutWindow/index';



webix.protoUI(componentLayout, webix.Movable, webix.DragItem, webix.ui.
   abslayout);
```

```
import BoundingClientRect from './BoundingClientRectInterface';

/* import grid from '../models/grid'; */

const drawShadow = (htmlView, left, top, grid): void => {
   const shadow = <HTMLElement>document.createElement('div');

   const viewRect = <BoundingClientRect>htmlView.children[0].
      getBoundingClientRect();
   shadow.className = 'element-shadow';

   shadow.style.top = `${top}px`;
   shadow.style.left = `${left}px`;
   shadow.style.width = `${viewRect.width - 1}px`;
   shadow.style.height = `${viewRect.height - 1}px`;
```

```
    shadow.style.position = 'relative';

    grid.appendChild(shadow);
};

const removeShadow = (): void => {
    let div = document.getElementsByClassName('element-shadow')[0];
    if (div && div.parentNode) {
        div.parentNode.removeChild(div);
    }
};

const drawShadowGrid = (htmlView: HTMLElement) => {
    const div = <HTMLElement>document.createElement('div');
    const viewRect = <BoundingClientRect>htmlView.getBoundingClientRect();
    div.className = 'grid-shadow';

    div.style.top = `${viewRect.top}px`;
    div.style.left = `${viewRect.left}px`;
    div.style.width = `${viewRect.width}px`;
    div.style.height = `${viewRect.height}px`;

    htmlView.appendChild(div);
};

const removeShadowGrid = (): void => {
    let div = document.getElementsByClassName('grid-shadow')[0];
    div.parentNode.removeChild(div);
};

export {
    drawShadow,
    drawShadowGrid,
    removeShadowGrid,
    removeShadow
};

import { stickToGrid } from './magneticGrid';
import { drawShadow, removeShadow } from './shadow';
import BoundingClientRect from './BoundingClientRectInterface';


webix.protoUI({
    name: 'elementMovable',
    move_setter(value: any): any {
        if (value) {
            this._move_admin = webix.clone(this._move_admin);
            this._move_admin.master = this;
            webix.DragControl.addDrag(this.$view, this._move_admin);
        }
        return value;
    },
    _move_admin: {
        $dragCreate(object, e): HTMLElement | boolean {
            let master = <any>webix.DragControl.getMaster(object).master;
```

```
        if ($$(e.target.getAttribute('view_id')) &&
            $$(e.target.getAttribute('view_id')).config.view === 'resizer') {
            return false;
        }

        if (master.config.move) {
            let offset = <{ x: number, y: number }>webix.html.offset(object);
            let pos = webix.html.pos(e);

            // TODO: need to get width of componentBar
            webix.DragControl.top = offset.y - pos.y;
            webix.DragControl.left = offset.x - pos.x;
            return webix.toNode(master.$view);
        }
        return false;
    },
    $dragDestroy(): void {
        const master = <BoundingClientRect>this.master.getChildViews()[0].
            $view.getBoundingClientRect();
        const grid = <any>this.master.getParentView();

        const gridBounds = <BoundingClientRect>this.master.getParentView()
            .$view
            .getBoundingClientRect();
        let view = <any>this.master;

        let diffX = <number>master.x - gridBounds.x;
        let diffY = <number>master.y - gridBounds.y;

        let { left, top } = <{ left: number, top: number }>stickToGrid(this.
            master, diffX, diffY, grid.config.separation);

        if (view._settings) {
            view._settings.top = top + 7;
            view._settings.left = left + 7;
        }

        webix.DragControl.top = 5;
        webix.DragControl.left = 5;

        view.resize();

        removeShadow();

        this.master.callEvent('onViewMoveEnd', []);
        let item = this.master.getChildViews()[0];

        item.config.top = this.master.config.top;
        item.config.left = this.master.config.left;

        const elem = <webix.ui.baselayout>view.getChildViews()[0];

        elem.callEvent('onBlur', []);
        elem.callEvent('onFocus', [elem]);
```

```
        },
        $dragPos(pos, e): void {
            this.master.callEvent('onViewMove', [pos, e]);

            const grid = <any>this.master.getParentView();
            const gridBounds = <BoundingClientRect>grid.$view.
                getBoundingClientRect();

            let context = webix.DragControl.getContext();
            let control = context.source;

            pos.x = e.clientX - gridBounds.x;
            control.style.left = pos.x;
            pos.y = e.clientY - gridBounds.y;
            control.style.top = pos.y;

            const master = <BoundingClientRect>this.master.getChildViews()[0].
                $view.getBoundingClientRect();

            let diffX = <number>master.left - gridBounds.x;
            let diffY = <number>master.top - gridBounds.y;

            let { left, top } = <{ left: number, top: number }>stickToGrid(this.
                master, diffX, diffY, grid.config.separation);

            let shadowRect = <HTMLElement>document.getElementsByClassName('
                element-shadow')[0];

            if (shadowRect) {
                removeShadow();
            }

            if (!shadowRect
                || (shadowRect.getBoundingClientRect().left !== left
                    && shadowRect.getBoundingClientRect().top !== top)) {
                drawShadow(this.master.$view, left + 10, top + 10, grid.getNode())
                    ;
            }
        }
    }
}, webix.ui.layout, webix.Movable);

import getGrid from '../helpers/getGrid';
import ComponentsLayout from '../interfaces/ComponentsLayout';
import BaseLayout from '../interfaces/BaseLayout';

const horizontalLayoutConfig: BaseLayout = {
    name: 'AppOrchid-Horizontal-Layout',
    defaults: {
        isEmpty: true,
        type: 'space',
        css: 'AO_Layout',
        template: '<div class="iii">Horizontal-Layout</div>',
    },
    $init(config): void {
```

```
      const grid = <ComponentsLayout>getGrid(this);

      if (typeof config.cols === 'undefined') {
         config.cols = [
            {
               template: '<div class="iii">Horizontal-Layout</div>',
               type: 'clean',
               height: grid.config.separation * 3,
               width: grid.config.separation * 6 > 220
                  ? grid.config.separation * 3
                  : grid.config.separation * 6
            }
         ];
      }
      webix.DragControl.addDrop(this.$view, this, true);
   }
};

export default horizontalLayoutConfig;
```

```
import '../baselayout';
import horizontalLayoutConfig from './config';

webix.protoUI(horizontalLayoutConfig, (<any>webix.ui).AppOrchidBaseLayout);
```

```
import GridLayout from '../interfaces/GridLayout';
import DragContext from '../interfaces/subinterfaces/DragContext';

import callWindowElementInsert from './dropComponentWindow/config';
import getGrid from '../helpers/getGrid';
/* import {getConfigId} from '../../componentsLayout/helpers/id'; */
import {addSelectedCss} from '../../../helpers/highlighting';

const gridlayoutConfig: GridLayout = {
   name: 'AppOrchid-Gridlayout',
   defaults: {
      isEmpty: true,
      type: 'space',
      autoplace: false
   },
   dropOptionsWindowId: 'AppOrchid-GridLayout-Drop-Options',
   $init() {
      webix.DragControl.addDrop(this.$view, this, true);
      webix.html.addCss(this.$view, 'gridlayout');
   },
   $drop() {
      let dnd: DragContext = webix.DragControl.getContext();
      if (dnd.from.config) {
         const grid = <webix.ui.gridlayout>getGrid(this);

         let elementConfig: object = webix.copy(dnd.from.getItem(dnd.source
            [0]).config);

         const gridElementConfig = {
            ...elementConfig,
            id: webix.uid(),
```

```
            parentId: /* getConfigId */(this.config.id)
         };

         let promise = new Promise((resolve, reject) => {
            callWindowElementInsert(resolve, reject);
         });

         promise
            .then((componentOptions) => {
               const resultConfig = {...gridElementConfig, ...componentOptions
                  };
               const elementId: object = this._addElement(resultConfig);
               grid.callEvent('componentAdded', [elementId]);
            })
            .catch(err => webix.message({
               text: err,
               type: 'error',
               expire: 2000
            }));
      }
   },
   _addElement(config) {
      webix.html.removeCss(<HTMLElement>this.$view, 'gridlayout');

      config.on = {
         onFocus(view) {
            const grid = <webix.ui.gridlayout>getGrid(view);

            addSelectedCss(view);

            grid.callEvent('onMovableElementSelect', [view]);
         }
      };
      /* const grid = getGrid(this);

      let thisFromCollection = grid.getSerializedCollection().find((element)
         => {
         if (element.id === this.config.id) {
            return element;
         }
      });

      console.log(thisFromCollection); */

      /* this.getChildViews().forEach((element) => {
         this.removeView(element);
      }); */

      /* webix.ui(thisFromCollection, this.getParentView()); */

      return <webix.ui.layout>this.addView(config);
   }
}

export default gridlayoutConfig;
```