МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра информационных технологий

КУРСОВАЯ РАБОТА

РАЗРАБОТКА ВЕБ ВЕРСИИ ТАБЛИЧНОГО РЕДАКТОРА

Работу выполнил _____ Е.А. Иванов
(подпись)

Направление подготовки 01.03.02 Прикладная математика и информатика

Направленность Программирование и информационные технологии

Научный руководитель
канд. физ.-мат. наук, доц. _____ Е.П. Лукащик
(подпись)

Нормоконтролер
ст.преп. _____А.В. Харченко
(подпись)

Краснодар
2022

РЕФЕРАТ

Курсовая работа содержит 30 страниц, 8 рисунков, 6 источников.

JAVASCRIPT, SASS, HTML, WEBPACK, BABEL, ТАБЛИЧНЫЙ РЕДАКТОР

Целью исследования является создание веб приложения – версии табличного редактора.

В основе приложения лежит клиент-серверная модель. При разработке клиентской части использовался язык JavaScript.

В результате была создана программная реализация табличного редактора со следующим функционалом:

- структурирование информации;
- удобное формирование информаций базы;
- возможность использовать формулы;
- удобное перемещение по ячейкам;
- обработка результатов экспериментов;
- работа над большими наборами данных;
- произведение однотипных расчётов.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В повседневной жизни человек постоянно использует таблицы: дневник в школе, расписание электричек, расписание занятий. Персональный компьютер расширяет возможности использования таблиц за счёт того, что позволяет не только представлять данные в электронном виде, но и обрабатывать их. Класс программного обеспечения, используемый для этой цели, называется табличными процессорами или электронными таблицами.

Основное назначение табличных процессоров – обработка таблично организованной информации, проведение расчётов на её основе и обеспечение визуального представления хранимых данных и результатов их обработки в виде графиков, диаграмм. Табличный процессор или электронная таблица – это интерактивная система обработки данных, в основе которой лежит двухмерная таблица. Ячейки таблицы могут содержать числа, строки или формулы, задающие зависимость ячейки от других ячеек. Пользователь может просматривать, задавать и изменять значение ячеек. Изменение значение ячейки ведет к немедленному изменению значений зависящих от нее ячеек.

Табличные процессоры обеспечивают также задание формата изображения, поиск, сортировку. Применение электронных таблиц упрощает работу с данными и позволяет получать результаты без проведения расчётов вручную. Расчёт по заданным формулам выполняется автоматически. Изменение содержимого, какой-либо ячейки приводит к перерасчёту значений всех ячеек, которые связаны с ней формульными отношениями. Электронные таблицы используются во всех сферах человеческой деятельности, но особо широко используются для проведения экономических и бухгалтерских расчётов.

Цель курсовой работы – создание табличного редактора для браузера, изучение языка программирования JavaScript и его возможностей в браузере.

1 Теоретические сведения и используемые технологии

1.1 Табличные и текстовые редакторы

Текстовый редактор — это программа, используемая специально для ввода и редактирования текстовых данных.

Этими данными может быть программа, или какой-либо документ, или же книга. Редактируемый текст выводится на экран, и пользователь в диалоговом режиме вносит в него свои изменения.

Текстовые редакторы могут обеспечивать выполнение разнообразных функций, а именно:

1) редактирование строк текста;
2) использование различных шрифтов;
3) копирование и перенос части текста с одного места на другое или из одного документа в другой;
4) контекстный поиск и замену частей текста;
5) задание произвольных межстрочных промежутков;
6) автоматический перенос слов на новую строку;
7) автоматическую нумерацию страниц;
8) обработку и нумерацию сносок;
9) выравнивание краев абзаца;
10) создание таблиц и построение диаграмм;
11) проверку правописания слов и подбор синонимов;
12) построение оглавлений и предметных указателей;
13) распечатку подготовленного текста на принтере в нужном числе экземпляров.

Табличное представление позволяет быстро осуществить расчеты над большими объемами данных. Если расчеты выполняются однократно, то оправдано использование калькулятора. Однако многократное выполнение расчетов вызывает утомление и раздражение,

что приводит к появлению ошибок и необходимости перерасчета, сопряженного с материальными и временными затратами.

Поскольку персональные компьютеры служат для выполнения многократно повторяющихся однообразных действий, то естественным стало использование при организации расчетов *электронных таблиц.* Пользователю предоставляется возможность размещать данные в таблице на экране монитора, являющейся аналогом таблицы на бумаге, и использовать для их обработки встроенные функции. Электронные таблицы (Spreadsheets — расширенные таблицы), или *табличные процессоры,* являются универсальными средствами для автоматизации расчетов над большими объемами табличных данных.

Электронная таблица создается и хранится в памяти компьютера. В дальнейшем ее возможно изменять, просматривать, сохранять на магнитном носителе, а также выводить копию на печать. На экране дисплея электронная таблица отображается в виде двухмерной матрицы, состоящей из столбцов и строк, на пересечении которых располагаются *ячейки (клетки).* В зависимости от используемого табличного процессора размер матрицы различен. Для обращения к содержимому ячейки используется однозначно определяющий ее идентификатор. В качестве такого идентификатора выступает *адрес* (номер столбца и номер строки, на пересечении которых располагается ячейка). В ячейки Электронной таблицы вводятся числа, текст, формулы или гиперссылки. Для задания текущей ячейки таблицы используется специальный *указатель ячейки (табличный курсор),* который имеет вид рамки. Ввод и редактирование данных осуществляются пользователем в текущую ячейку.

Концепция электронной таблицы, впервые реализованная для компьютера фирмы Apple, оказалась удачной, и в течение нескольких лет был выпущен ряд программных средств этого класса (семейства yisicalc, Supercalc, Multiplan). Электронные таблицы стали одним ив основных

компонентов интегрированных пакетов программ Works, Symphony. Большое влияние на развитие программных средств этого класса оказала разработка пакета Lotus 1-2-3 фирмы Lotus Development. Этот пакет благодаря своим функциональным возможностям и скорости обработки долгое время являлся эталоном для аналогичных программных продуктов.

Разработчики табличных процессоров ориентируются на следующие критерии: расширение функциональных возможностей, увеличение скорости обработки, обеспечение простоты изучения и удобства использования. Современные табличные процессоры содержат средства для работы с текстами, таблицами, графикой, гиперсвязями, а также дополнения для моделирования, анализа и прогнозирования.

На практике электронные таблицы получили широкое распространение при экономических расчетах. Это обусловлено тем, что решение большинства экономических задач связано с обработкой табличных документов и результаты решения ряда задач следует представлять в табличной форме.

На рынке программных продуктов наиболее популярными представителями этого класса являются табличные процессоры различных версий *Lotus 1-2-3* фирмы Lotus Development Inc., *Quattro Pro* фирмы Novell и *Excel* корпорации Microsoft. Эти продукты являются компонентами соответствующих офисных пакетов — Lotus SmartSuit, Perfect Office и Microsoft Office. Функциональные возможности табличного процессора обеспечивают его широкое использование для финансовой обработки данных, научных и инженерно-технических расчетов, автоматизации учетно-контрольной деятельности, эффективной обработки больших объемов информации, заданных в табличном виде.

Возможности табличных редакторов различны — от программ, предназначенных для подготовки небольших документов простой структуры, до программ для набора, оформления и полной подготовки к типографскому изданию книг и журналов (издательские системы).

Издательские системы незаменимы для компьютерной верстки и графики. Они значительно облегчают работу с многостраничными документами, имеют возможность автоматической разбивки текста на страницы, расстановки номеров страниц, создания заголовков и т. д. Создание макетов любых изданий — от рекламных листков до многостраничных книг, журналов и Web-страниц — становится доступным даже для новичков.

Табличный редактор (процессор) — это комплекс взаимосвязанных программ, предназначенный для обработки электронных таблиц. Электронная таблица – это компьютерный эквивалент обычной таблицы, состоящей из строк и граф, на пересечении которых располагаются клетки, содержащие числовую информацию, формулы или текст.

Значение в числовой клетке таблицы может быть либо записано, либо рассчитано по соответствующей формуле; в формуле могут присутствовать обращения к другим клеткам.

Каждый раз при изменении значения в клетке таблицы в результате записи в нее нового значения с клавиатуры пересчитываются также значения во всех тех клетках, в которых стоят величины, зависящие от данной клетки.

Графам и строкам можно присваивать наименования. Экран монитора трактуется как окно, через которое можно рассматривать таблицу целиком или по частям.

Табличные процессоры представляют собой удобное средство для проведения бухгалтерских и статистических расчетов. В каждом пакете имеются сотни встроенных математических функций и алгоритмов статистической обработки данных. Кроме того, имеются мощные средства для связи таблиц между собой, создания и редактирования электронных баз данных.

Специальные средства позволяют автоматически получать и распечатывать настраиваемые отчеты с использованием десятков различных типов таблиц, графиков, диаграмм, снабжать их комментариями и графическими иллюстрациями. Табличные процессоры имеют встроенную

справочную систему, предоставляющую пользователю информацию по конкретным командам меню и другие справочные данные. Многомерные таблицы позволяют быстро делать выборки в базе данных по любому критерию.

Самые популярные табличные процессоры Microsoft Excel и Lotus 1-2 3.

В Microsoft Excel автоматизированы многие рутинные операции, специальные шаблоны помогают создавать отчеты, импортировать данные и многое другое.

Lotus 1-2-3 — профессиональный процессор электронных таблиц. Широкие графические возможности и удобный интерфейс пакета позволяют быстро ориентироваться в нем. С его помощью можно создать любой финансовый документ, отчет для бухгалтерии, составить бюджет, а затем разместить эти документы в базах данных.

Табличный процессор есть, в сущности, совмещение текстового редактора с электронным калькулятором.

Его можно применять как большой математический калькулятор, когда нужно быстро обсчитать табличные данные по некоторым формулам.

Он позволяет составить таблицу, включить в нее формулы для обсчета табличных данных, произвести вычисления по этим формулам, записать полученную таблицу на диск и использовать затем многократно, изменяя лишь данные.

Современные табличные процессоры обеспечивают представление табличных данных в графической форме – в виде графиков и диаграмм. Причем позволяют оформлять их, как и таблицы, весьма профессионально и красочно, не затрачивая особых усилий. В этом отношении они не заменимы.

Но следует отметить, что табличный процессор не является универсальным вычислительным средством, как считают многие. Он способен решать довольно ограниченный круг стандартных задач, в основном экономического характера.

Следует развеять и другое устоявшееся заблуждение. Считается большим достоинством табличного процессора тот факт, что «решение задачи с его помощью не требует участия программиста. Пользователь самостоятельно решает свою задачу от начала до конца». То есть, видимо, имеется в виду отсутствие необходимости в составлении программы решения задачи.

Однако при решении задачи с помощью табличного процессора также составляется программа: те формулы, которые «пользователь вводит самостоятельно», и есть программа, только по структуре она линейная, и потому создание ее не вызывает затруднений.

Работа с табличным процессором требует и составления алгоритма, но здесь он обычно называется схему решения. Для простых задач схема очень проста и однотипна, но составить подобную схему для более или менее серьезной задачи – дело очень не простое, может быть, даже более сложное, чем составление соответствующего алгоритма для программы на Бейсике – требуются высокая квалификация в этой области и хорошее знание табличного процессора, всех его возможностей.

Требуется здесь выполнять и математическую постановку задач (нестандартных), но обо всем этом почему-то обычно забывают. И еще: работать с табличным процессором должен работать профессионал, тогда как с программой на Бейсике или другом языке может работать конечный пользователь, т.е. любой гражданин России.

Табличный редактор позволяет решать стандартный для данного вида круг задач, причем с высокой степенью комфорта для пользователя. Кроме того, он содержит средства, значительно расширяющие возможности его как табличного процессора.

Так, он дает возможность работать с небольшими базами данных, когда требуется несложная выборка данных по определенному критерию; содержит встроенные средства макропрограммирования, позволяющие объединять возможности табличного процессора и языков программирования и

разрабатывать законченные программы для конечных пользователей (несложные программы бухгалтерского, складского учёта). Именно эти средства резко повышают ценность табличных редакторов и расширяют область его применения.

1.2 Используемые технологии и сведения о них

При реализации данного проекта были использованы следующие технологии: Sass, JavaScript, HTML, Webpack, Babel.

Sass — это метаязык на основе CSS, предназначенный для увеличения уровня абстракции CSS-кода и упрощения файлов каскадных таблиц стилей.

JavaScript – мультипарадигменный, динамический, слабо и неявно типизированный язык программирования.

HTML – язык гипертекстовой разметки.

Webpack – сборщик модулей JavaScript с открытым исходным кодом.

Babel - транскомпилятор JavaScript с открытым исходным кодом, который в основном используется для преобразования кода ECMAScript 2015+ в обратно совместимую версию JavaScript, которая может выполняться старыми движками JavaScript.

1.3 Реализованные технологии и паттерны

1.3.1 Паттерн Observer

В данной курсовой работе используются такой поведенческий паттерн как Observer. Паттерн Observer один из популярнейших паттернов, используемый в сайтостроении и встречающийся практически каждом приложений на JavaScript.

Наблюдатель (англ. Observer) — поведенческий шаблон проектирования. Также известен как «подчинённые» (Dependents).

Создает механизм у класса, который позволяет получать экземпляру объекта этого класса оповещения от других объектов об изменении их состояния, тем самым наблюдая за ними.

В каких случаях используется Наблюдатель?

1) Если один объект должен передавать сообщения другим объектам, но при этом он не может или не должен знать об их внутреннем устройстве;

2) В случае если при изменении одного объекта необходимо изменять другие объекты;

3) Для предотвращения сильных связей между объектами системы;

4) Для наблюдения за состоянием определенных объектов системы.

Наблюдаемый (observable) объект, а точнее субъект или subject, должен предоставлять интерфейс для регистрации и дерегистрации наблюдателей (listeners).

Ну а сами наблюдатели должны как минимум обладать открытым методом через который и будет происходить оповещение об изменении состояния субъекта.

Данный шаблон часто применяют в ситуациях, в которых отправителя сообщений не интересует, что делают получатели с предоставленной им информацией.

Паттерн Observer определяет зависимость "один-ко-многим" между объектами так, что при изменении состояния одного объекта все зависящие от него объекты уведомляются и обновляются автоматически.

Паттерн Observer инкапсулирует главный (независимый) компонент в абстракцию Subject и изменяемые (зависимые) компоненты в иерархию Observer.

Паттерн Observer определяет часть "View" в модели Model-View-Controller (MVC).

Экземпляр (объект) содержит коллекцию объектов (наблюдателей) и уведомляет их обо всех случаях, когда происходят какие-либо изменения их состояния. Этот позволяет делать связи один ко многим между компонентами.

Мы создаем один Observer и потом в нескольких местах подписываемся на события этого Observer с помощью предоставленного метода on(), который принимает событие и функцию-колбэк, которая будет вызвана при тригере данного события в методу trigger(). Компоненты, у которых появилась нужная информация, вызывают метод trigger(), выполняющий все переданные функции-колбэки для нужного события с переданными данными.

```
export class Observer {
  constructor() {
    this.listeners = {}
  }

  trigger(event, ...args) {
    if (!Array.isArray(this.listeners[event])) {
      return false
    }
    this.listeners[event].forEach(listener => {
      listener(...args)
    })

    return true
  }

  on(event, fn) {
    this.listeners[event] = this.listeners[event] || []
    this.listeners[event].push(fn)
    return () => {
      this.listeners[event] =
        this.listeners[event].filter(listener => listener !== fn)
    }
  }
}
```

Рисунок 1 – Реализация паттерна Observer

1.3.2 Собственная реализация jQuery

Во время разработки на JavaScript часто приходиться использовать довольно громоздкие конструкции из кода, чтобы выполнить простую функцию. Для упрощения работы существует библиотека jQuery, которая представляет собой набор JavaScript функций. Особенно удобно использовать jQuery для создания интерактивных сайтов. Именно поэтому в процессе написания курсовой работы было также реализовано собственное решение, которое является альтернативой jQuery.

Класс Dom представляет обертку для удобного взаимодействия с Dom-деревом. Этот класс предоставляет методы для обработки Dom-узлов: применение к ним стилей, фокусировка, получение координат, данных и стилей этого узла, добавления и удаление атрибутов, доступ к соседним узлам, очищение контента внутри узлов. Она также позволяет использовать chaining для комфортного использования. Данная обертка выносит логику взаимодействия с Dom-деревом из компонентов, которые не должны знать детали управления, и предлагает удобный интерфейс взаимодействия с узлами приложения.

Данная реализация позволяет писать код более компактно, значительно упрощает написание многих вещей, например, таких как манипулирование DOM элементами, обработку событий, добавление эффектов анимации на страницу.

1.3.3 Собственная реализация Redux

Также была написана собственная реализация Redux. Redux - state manager, библиотека, предназначенная для управления состоянием приложения.

В Redux общее состояние приложения представлено одним объектом JavaScript — state (состояние) или state tree (дерево состояний). Неизменяемое дерево состояний доступно только для чтения, изменить ничего напрямую нельзя. Изменения возможны только при отправке action (действия).

Действие (action) — это JavaScript-объект, который лаконично описывает суть изменения.

Генераторы действий (actions creators) — это функции, создающие действия.

Редуктор (reducer) — это чистая функция, которая вычисляет следующее состояние дерева на основании его предыдущего состояния и применяемого действия.

Чистая функция работает независимо от состояния программы и выдаёт выходное значение, принимая входное и не меняя ничего в нём и в остальной программе. Получается, что редуктор возвращает совершенно новый объект дерева состояний, которым заменяется предыдущий.

В данной работе нам необходимо сохранять информацию, которая относится непосредственно к состоянию приложения: выбранные для ячейки шрифты, стили, название таблиц, дата открытия таблиц, состояние строк и столбцов. В работе присутствует функции, ответственные за данные взаимодействия.

```
export function createStore(rootReducer, initialState = {}) {
  let state = rootReducer({...initialState}, {type: '__INIT__'})
  let listeners = []

  return {
    subscribe(fn) {
      listeners.push(fn)
      return {
        unsubscribe() {
          listeners = listeners.filter(listener => listener !== fn)
        }
      }
    },
    dispatch(action) {
      state = rootReducer(state, action)
      listeners.forEach(listener => listener(state))
    },
    getState() {
      return JSON.parse(JSON.stringify(state))
    }
  }
}
```

Рисунок 2 – Реализация функции для отслеживания состояния приложения

Функция createStore принимает в качестве параметров rootReducer и initialState, являющийся по умолчанию пустым объектом. rootReducer – функция, ответственная за все изменения состояния приложения. initialState – начальное состояние приложения. Данная функция возвращает методы подписки на изменения состояния, получения состояния и отправки action для изменения состояния. Метод subscribe() представляет собой замыкание для

переданной в нее функции fn. Функция может быть динамически создана, скопирована в другую переменную или передана как аргумент другой функции и позже вызвана из совершенно другого места. В JavaScript у каждой выполняемой функции, блока кода и скрипта есть связанный с ними внутренний (скрытый) объект, называемый лексическим окружением. Объект лексического окружения состоит из двух частей:

1) environment record – объект, в котором как свойства хранятся все локальные переменные (а также некоторая другая информация, такая как значение this);

2) ссылка на внешнее лексическое окружение – то есть то, которое соответствует коду снаружи (снаружи от текущих фигурных скобок);

Лексическая область видимости — это статическая область в JavaScript, имеющая прямое отношение к доступу к переменным, функциям и объектам, основываясь на их расположении в коде.

Каждый раз, когда движок JavaScript создаёт контекст выполнения, чтобы выполнить функцию или глобальный код, он также создаёт новое лексическое окружение, чтобы хранить переменную, определенную в этой функции во время её выполнения.

Лексическое окружение это структура данных, которая хранит информацию по идентификаторам переменных. Тут идентификатор обозначает имя переменных/функций, а переменная настоящий объект или примитивное значение.

У лексического окружения есть два компонента: (1) запись в окружении и (2) отсылка к внешнему окружению.

1) запись в окружении это место хранятся объявления переменной или функции;

2) отсылка к внешнему окружению означает то, что у него есть доступ к внешнему (родительскому) лексическому окружению;

Когда движок JavaScript создаёт глобальный контекст выполнения, чтобы выполнить глобальный код, он также создаёт новое лексическое

окружение, чтобы хранить переменные и функции, определенные в глобальной области видимости.

Когда движок создаёт контекст выполнения для функции, он также создаёт лексическое окружение для хранения переменных, объявленных в этой функции во время выполнения.

Внешнее лексическое окружение функции указывается в глобальном лексическом окружении, потому что функция окружена глобальной областью видимости в исходном коде.

Контекст выполнения — это абстрактная среда, в которой JavaScript код оценивается и выполняется. Когда выполняется “глобальный” код, он выполняется внутри глобального контекста выполнения, а код функции выполняется внутри контекста выполнения функции.

Тут может быть только один запущенный контекст выполнения (JavaScript это однопоточный язык), который управляется стеком запросов.

Стек выполнения это стек с принципом *LIFO (Последний вошёл, первый вышел)*, в котором элементы могут быть добавлены или удалены только сверху стека.

Запущенный контекст выполнения будет всегда сверху стека и когда запущенная функция завершится, её контекст выполнения выкинется из стека, запустив контекст выполнения, который стоит ниже в очереди.

"Переменная" – это просто свойство специального внутреннего объекта: Environment Record. «Получить или изменить переменную», означает, «получить или изменить свойство этого объекта».

Замыкание — это функция у которой есть доступ к своей внешней функции по области видимости, даже после того, как внешняя функция прекратилась. Это говорит о том, что замыкание может запоминать и получать доступ к переменным, и аргументам своей внешней функции, даже после того, как та прекратит выполнение.

Таким образом метод subscribe() возвращает функции отписки от состояния приложения в случае, если подписка больше не нужна, и для

предотвращения утечек памяти. Метод dispatch получает на вход action – событие, которое содержит в себе изменения состояния, и передает этот action в rootReducer для получения обновленного состояния. Далее все подписанные компоненты получают обновленное состояния и отрисовывают данные пользователю. Метод getState возвращает состояние приложения. rootReducer – функция, принимающая определенные объекты – action, содержащие тип изменения и необходимые данные. Данный rootReducer поддерживает типы action: изменение размеров таблицы, изменение текста, стилей, заголовка таблицы, выбор ячеек, даты открытия таблицы, то есть всю необходимую информацию для изначальной отрисовки пользовательской таблицы. RootReducer в зависимости от типа action будет изменять состояние приложения, основываясь на предыдущем состоянии и пришедших ему данных для дальнейшего внесения в состояние. Объект состояния так же, как и в Redux, является иммутабельным для предотвращения нежелательных побочный эффектов.

2 Программная реализация и функционал

2.1 Общий принцип работы

Приложение представляет собой сайт, который строит две страницы в соответствии с шаблонами. Первый шаблон – страница выбора и создания таблиц. Второй шаблон – страница таблицы.

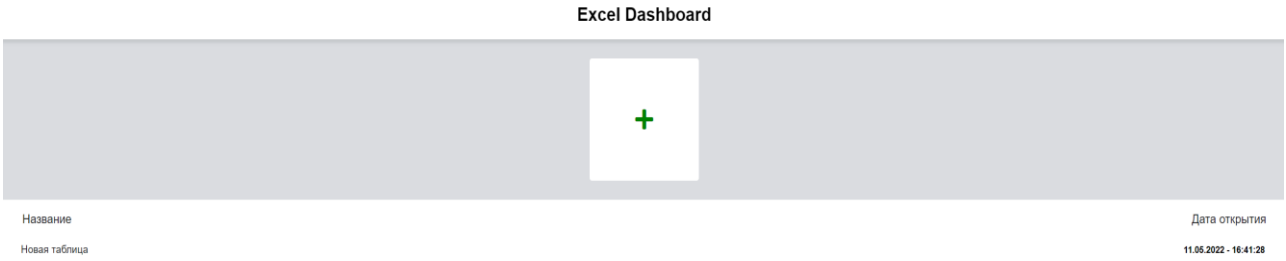

Рисунок 3 – Страница создания таблиц

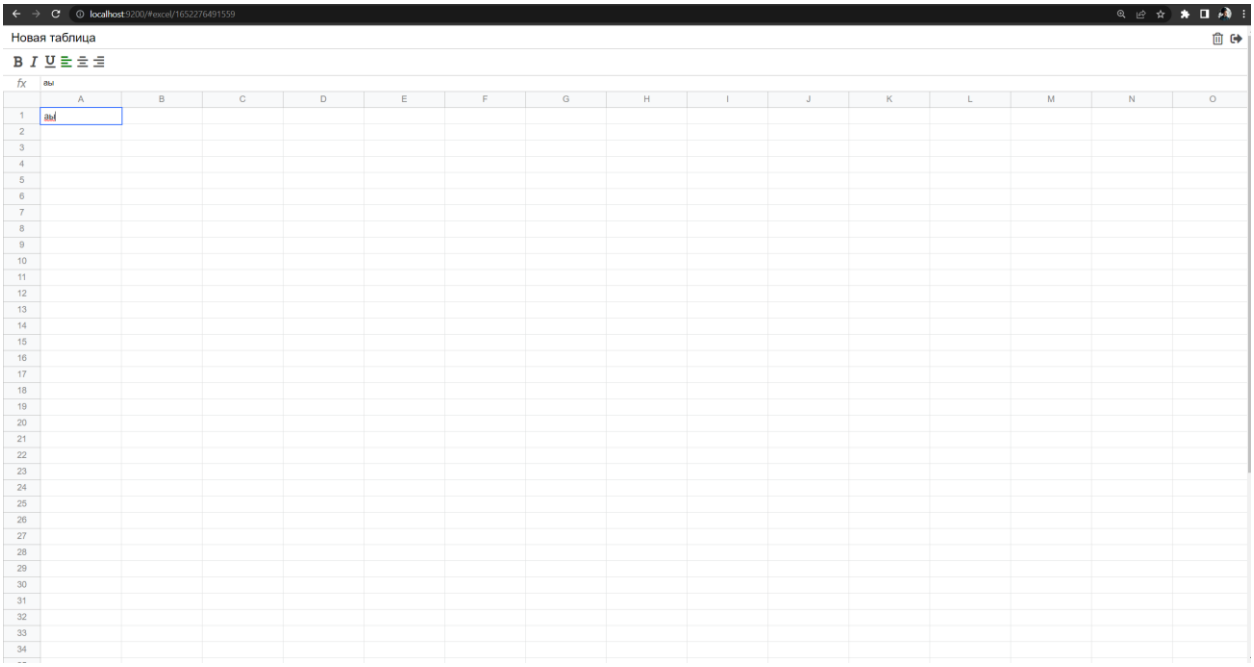

Рисунок 4 – Страница таблицы

Страницы таблицы представляет собой 4 компонента: Header, Toolbar, Formula и Table.

Все эти компоненты наследуются от основного компонента ExcelComponent, который решает проблему дублирования кода. В этом компоненте-обертке находятся все методы, необходимые дочерним компонентам: подписки на Observer и их отслеживания и состояния приложения, проверка на прослушивание определенным компонентом определенного события Observer-а. ExcelComponent в свою очередь наследуется от класса-обертки DomListener, который следит за методами дочерних классов и осуществляет добавление и удаление (в целях избежания утечек памяти) слушателей на dom-узлы приложения. Слушатель представляет собой метод addEventListener. Этот метод регистрирует обработчик события для целевого объекта (eventTarget), для которого он будет вызываться при возникновении события. Целевым объектом может быть объект Element, Document, Window или любой другой объект, поддерживающий события, например, такой как XMLHttpRequest. При инициализации DomListener получает массив событий, на которые необходимо повесить слушатели событий. Проходя по массиву, необходимо понять, какую функцию нужно вызывать при некотором событии. В данной работе введена следующая схема: методы для определенных событий содержатся в классах-компонентах, в которых и происходит событие. Название методов исходят из того, какое событие необходимо обработать. Если это click, то соответствующий метод будет называться onClick. Таким образом DomListener знает, какие методы компонентов необходимо вызывать при определенном событии. Ему остается лишь повесить слушатель события на данное событие и назначить ему метод по схеме описанной выше. Нужное нам событие приходит от самого браузера, а именно веб API Event. Интерфейс Event представляет собой любое событие, которое происходит в DOM; некоторые из них генерируемые пользователем (клик мышью или

нажатие клавиши на клавиатуре), а некоторые - генерируемые API (события, обозначающие завершение процесса анимации, приостановка видео и т.д.).

При обработке события браузер автоматически передает в функцию обработчика в качестве параметра объект Event, который инкапсулирует всю информацию о событии. И с помощью его свойств мы можем получить эту информацию: bubbles: возвращает true , если событие является восходящим.

Существует много типов событий, некоторые из них используют интерфейсы, базирующиеся на главном интерфейсе Event. Event содержит общие свойства и методы для всех событий.

В каждом компоненте в конструкторе определен массив событий, на которые должны быть повешены слушатели событий. Каждый компонент передает свой массив на самый верх цепочки – в DomListener, который осуществляет добавление слушателей на заданные события. В каждом компоненте присутствуют методы для этих событий (функции-колбэки, которые должны сработать по достижение того или иного события), представленные в особой форме для удобного расширения приложения. Если событие называется click, то функция-колбэк должна иметь вид onClick. При уничтожении компонентов каждый из них активирует функцию removeDomListener в классе DomListener, которая имеет уже все нужные массивы событий и использует функцию getMethodName для приведения ее к функции-колбэку, чтобы эффективно осуществить удаление всех слушателей для компонента. Это необходимо во избежание различных утечек памяти. Утечка памяти — память, которая больше не требуется приложению, но по какой-то причине не возвращается операционной системе или пулу доступной памяти. Языки программирования используют разные подходы, снижающие риск возникновения утечек памяти, однако сама задача о том, понадобится ли ещё определенный фрагмент памяти или нет, алгоритмически неразрешима. Иными словами, только разработчик может определить, возможно ли вернуть определенный фрагмент памяти операционной системе. Управление памятью в языках программирования делится на ручное и автоматическое. Первый тип

предоставляет разработчику набор инструментов, помогающих напрямую взаимодействовать с памятью. Во втором существует специальный процесс, называемый «сборщиком мусора», вызываемый автоматически и удаляющий память. Движок JavaScript имеет сборщик мусора, который периодически проверяет, какие из выделенных приложению фрагментов памяти остаются достижимы из различных частей приложения. Хотя сборщики мусора и полезны, у них есть свои недостатки, одним из которых является недетерминированность. Это значит, что сборщики мусора непредсказуемы — обычно невозможно определить, когда будет произведена сборка мусора. Как следствие, иногда программа занимает больше памяти, чем требуется. Также могут наблюдаться кратковременные паузы, что будет особенно заметно в быстро реагирующих на действия программах.

Недетерминированность означает, что мы не можем точно сказать, когда будет произведена сборка мусора, однако большинство реализаций сборщиков мусора имеют сходное поведение. Если не производится выделений памяти, сборщик мусора никак себя не проявляет.

Поэтому полагаться на сборщика мусора в удалении слушателей DOM-элементов не стоит, потому что он может решить, что слушатель все еще нужен, и не станет его удалять. Именно поэтому в классе DomListener реализован метод removeDomListener, который явно удаляет всех ненужных приложению слушателей событий.

Навигация реализована с помощью класса Router, который отслеживает изменение хеша и подгружает нужную страницу (ExcelPage или DashboardPage) с нужными параметрами.

ExcelPage создает экземпляр состояния приложения и регистрирует нужные компоненты для рендера. После успешного рендера все компоненты инициализируются и получает нужный функционал для работы.

DashboardPage получает все необходимые записи таблиц и успешно рендерит их на странице.

2.2 Работа компонентов

2.2.1 Компонент Header

Компонент Header содержит 3 активности: изменение названия таблицы, удаление таблицы и выход на страницу создания таблиц. Header подписан на изменения состояния приложения для отслеживания названия таблицы и имеет 2 типа прослушивания событий клик и ввод. Ввод – является в данном случае функцией отложенного действия, чтобы не вызывать функцию изменения состояния на каждое нажатие пользователя. Задержка в 300 миллисекунд дает возможность сократить вызовы трудоемких функций. Для взаимодействия с полем ввода используется собственная реализация библиотеки jQuery. Удаление таблицы происходит при нажатии соответствующей кнопки удаления. Определение нажатия по определенной кнопки осуществляется путем просмотра атрибутов кнопки. Далее пользователю показывается диалоговое окно с помощью встроенной функции confirm для подтверждения удаления. Определение нужной таблицы для удаления основывается на уникальном айди каждой таблицы, который находится в url адресе таблицы. Параметры и навигация осуществляется также с помощью встроенных методов window.location, который обернут в удобный компонент ActiveRoute со статическими методами.

2.2.2 Компонент Toolbar

Компонент Toolbar содержит 6 кнопок для редактирования текста ячеек.

Клик по одной из кнопок компонента триггерит событие, которое передается в Observer с данными для дальнейшей передачи определенной ячейке, на которой было произведено нажатие для применения выбранного пользователем стиля.

При выборе определенной ячейки должны отображаться активными конкретные кнопки. Таким образом компонент Toolbar использует состояние приложения для определение активных кнопок. При выборе конкретной ячейки изменяется состояние currentCell, на которое подписан компонент Toolbar, ведущее к перерендеру компонента Toolbar.

2.2.3 Компонент Formula

Компонент Formula отображает действия в конкретной ячейке.

При инициализации компонента получается доступ к dom-узлу, содержащему данный компонент, и осуществляется подписка на table:select, выбор ячейки в таблице для отображение в компоненте Formula соответствующих данных ячейки.

С помощью Observer компонент подписан на изменения данных в ячейках и отображает соответствующий текст при переключении между различными ячейками. При каждом открытии таблицы в формуле будет отображаться значение конкретной ячейки, которое она получит из состояния приложения. Пользователь может вводить данные прямо в формуле, и они будут так же отображаться в ячейках, так как ячейки подписаны через Observer на изменение формулы. Формула также поддерживает использование арифметических действий, причем в ячейке будет выводиться ответ выражения из формулы, а в самой формуле сохраняться введенное пользователем математическое выражение.

2.2.4 Компонент Table

Компонент Table является самым объемным в приложении, так как именно в нем содержится основная логика приложения.

Поскольку ячеек таблицы может быть огромное количество, то не представляется возможным вешать слушатель событий для каждой ячейки

(слишком неоптимальное решение). Таким образом для компонента Table реализовано делегирование событий. Именно всплытие и перехват событий позволяет реализовать этот один из самых важных приемов разработки. Когда на элементе происходит событие, обработчики сначала срабатывают на нём, потом на его родителе, затем выше и так далее, вверх по цепочке предков. Всплытие идёт с «целевого» элемента прямо наверх. По умолчанию событие будет всплывать до элемента <html>, а затем до объекта document, а иногда даже до window, вызывая все обработчики на своём пути. Но любой промежуточный обработчик может решить, что событие полностью обработано, и остановить всплытие. Для этого нужно вызвать метод event.stopPropagation(). Сама идея делегирования событий в том, что если у нас есть много элементов, события на которых нужно обрабатывать похожим образом, то вместо того, чтобы назначать обработчик каждому, мы ставим один обработчик на их общего предка. Из него можно получить целевой элемент event.target, понять на каком именно потомке произошло событие и обработать его. Таким образом нам не нужны тысячи слушателей событий, каждый из которых будет слушать свою отдельную ячейку. А нужен всего один слушатель, которым нужно навесить на родителя всех ячеек.

При инициализации компонента происходит выбор ячейки по умолчанию для ее выделения и фокуса, создаются подписки formula:input для отслеживания ввода в компоненте Formula и синхронного отображения этих же данных в ячейке таблицы, formula:done для фокуса и выделения заполненной через компонент Formula ячейки и toolbar:applyStyle для применения стилей ячейки или группы ячеек с последующим обновлением состояния стилей приложения.

При подготовке шаблона таблицы для последующего рендера на пользовательском экране необходимо собрать всю информацию о ячейках из состояния приложения. Для рендера каждой ячейки таблицы используется отдельная функция, принимающая строку, столбец, ширину, контент и объект стилей. Айди каждой клетки получается конкатенированием строки и столбца,

где находится ячейка. Применяются необходимые стили ячейки, задается ширина и высота. В состоянии приложения сохраняются только как-либо измененные ячейки. Ячейки, которые пользователь не трогал, в состоянии отсутствуют и рендерятся с дефолтными значениями.

Строки и столбцы таблицы можно увеличивать и уменьшать в размерах. Специальная функция resizeHandler принимает корневой элемент и элемент resizer, который показывает возможность действия. Для определения строки или столбца с помощью собственной реализации jQuery, метода closest мы определяем родителя resizer, строку или столбец, которые необходимо изменить по размерам. Далее на весь документ вешается прослушиватель на движение мыши, который определяет направление и координаты для смещения.

```
const delta = event.pageX - coords.right
value = coords.width + delta
$resizer.css( styles: {
  right: -delta + 'px'
})
```

Рисунок 5 – Вычисление новой ширины

Высчитывается delta = координаты события движения мыши по оси X – координаты правого края столбца. После чего ширина столбца увеличивается на соответствующую delta и применяются стили для отображения пользователю.

```
const delta = event.pageY - (coords.bottom + pageYOffset)
value = coords.height + delta
$resizer.css( styles: {
  bottom: -delta + 'px'
})
```

Рисунок 6 – Вычисление новой высоты

Для изменения размера строк delta = координаты события движения мыши по оси Y – (координаты нижней границы строки + пользовательская прокрутка страницы). После чего высота строки увеличивается на соответствующую delta и применяются стили для отображения пользователю.

Чтобы пользователь видел будущую границу столбца, отображается синяя полоска.

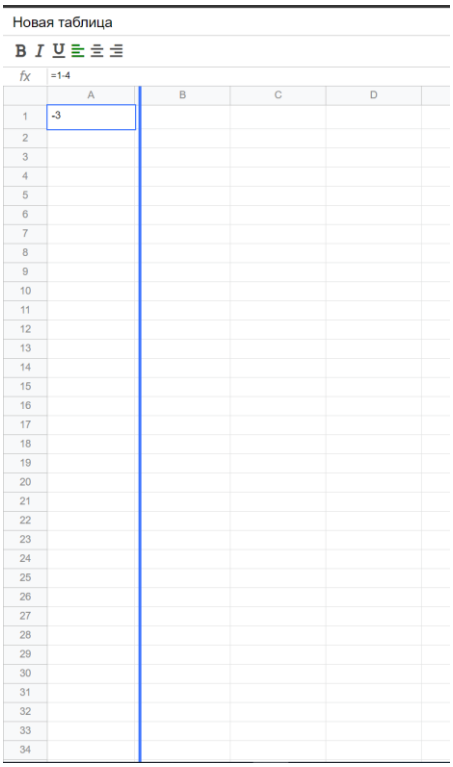

Рисунок 7 – Будущая граница расширяемого столбца

После того как пользователь отпустит кнопку мыши, все прослушиватели будут удалены, а граница столбца будет изменена.

При выборе конкретной ячейки таблицы происходит триггер события table:select, который обновляет данные в компоненте Toolbar в соответствии с данными в ячейке. Также вызывается метод dispatch для сохранения стилей и данных ячейки. Есть возможность выбирать группы ячеек для массового применения стилей. Данная функция активируется при зажатии кнопки Shift на клавиатуре и выборе двух граничных точек. Специальная функция selectedMatrix принимает две ячейки и определяет их айди. В матрице с выбранными граничными точками определяются все айдишники ячеек и складываются в массив для дальнейшего применения выбранных пользователем действий.

Также реализована возможность перемещения по таблице с использованием стрелок на клавиатуре. Функция onKeyDown принимает событие. Далее осуществляется проверка на совпадение нажатой кнопки с кнопками доступными для перемещения среди ячеек. После этого необходимо узнать, была ли зажата кнопка Shift, позволяющая выбрать матрицу ячеек. В случае выбора матрицы срабатывает функция selectedMatrix, описанная выше. В другом случае осуществляется переход в выбранную пользователем ячейку с помощью функции nextSelector, которая определяет по айдишникам следующую ячейку при перемещении с использованием стрелок, осуществляя проверку выхода за пределы таблицы.

2.2.5 Компонент Excel

Компонент Excel является сборщиком всех функционирующих компонентов, именно этот компонент вызывает инициализацию всех других компонентов, обновляет дату открытия для таблицы и подписывает компоненты на обновления состояния приложения. Отдельные компоненты содержат массив необходимых им данных из состояния всего приложения.

При обновлении определенных кусков состояния выполняется поиск компонентов, подписанных на эти изменения, и вызываются соответствующие функции-колбэки этих компонентов, которые вызывают перерендер компонентов.

```
export class StoreSubscriber {
  constructor(store) {
    this.store = store
    this.sub = null
    this.prevState = {}
  }

  subscribeComponents(components) {
    this.prevState = this.store.getState()
    this.sub = this.store.subscribe(state => {
      Object.keys(state).forEach(key => {
        if (!isEqual(this.prevState[key], state[key])) {
          components.forEach(component => {
            if (component.isWatching(key)) {
              const changes = {[key]: state[key]}
              component.storeChanged(changes)
            }
          })
        }
      })

      this.prevState = this.store.getState()
    })
  }
```

Рисунок 8 – Функция подписки на состояние приложения

А также именно этот компонент вызывает уничтожение всех компонентов для избежания утечки памяти.

ЗАКЛЮЧЕНИЕ

В курсовой работе были изучены особенности языка программирования JavaScript и разработан собственный табличный редактор.

В работе изучены особенности и применение Web API Event, предоставляющей огромный спектр событийных моделей браузера, обработка различных событий с использование всплытия, погружения и делегирования, основные концепции языка программирования JavaScript, особенности использования замыканий.

Практическая часть работы представлена браузерным табличным редактором, использующим поведенческий паттерн программирования Observer и собственные реализации библиотек jQuery и Redux.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Документация Sass: официальный сайт – URL: https://sass-scss.ru/documentation/ (дата обращения: 12.03.2022).

2 Документация JavaScript: официальный сайт – URL: https://developer.mozilla.org/ru/docs/Web/JavaScript (дата обращения 15.03.2022).

3 Справочник по HTML: официальный сайт – URL: http://htmlbook.ru/html (дата обращения 23.03.2022).

4 Руководство по CSS: официальный сайт – URL: https://developer.mozilla.org/ru/docs/Web/CSS/Reference (дата обращения 28.03.2022).

5 Документация Webpack: официальный сайт – URL: https://webpack.js.org/concepts/ (дата обращения 01.04.2022).

6 Документация Babel: официальный сайт – URL: https://babeljs.io/docs/en/ (дата обращения 16.04.2022).