|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатики, искусственного интеллекта и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОЙ РАБОТЕ***

***НА ТЕМУ:***

***Разработка базы данных для обеспечения***

***тренировок в фитнес-клубе***

Студент ИУ9-62Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Терентьева А. С.**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Домрачева А. Б.**

(Подпись, дата) (И.О.Фамилия)

*Москва, 2023 г.*

**ОГЛАВЛЕНИЕ**

ВВЕДЕНИЕ 3

1. Теоретические сведения 4

1.1. Диаграмма Ганта 4

1.2. Инструменты web-разработки 5

1.3. Векторная графика 6

1.4. Кривая Безье 9

2. Реализация 11

2.1. Реализация временной шкалы 11

2.2. Реализация отображения блоков задач 16

2.3. Реализация размещения дуг связей между блоками задач 17

2.4. Функциональные возможности приложения 24

3. Оценка визуализации и обзор аналогов 29

ЗАКЛЮЧЕНИЕ 31

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 32

ВВЕДЕНИЕ

Цель данной курсовой работы - разработать базу данных для фитнес-клуба, которая позволит вести учет тренировок пользователей и хранить статистику их показателей тела. База данных является важным инструментом для успешного функционирования фитнес-клуба по следующим причинам:

1. Отслеживание прогресса: база данных может хранить данные о весе, объеме талии, бедер и т.д., а также данные о тренировках, такие как количество повторений, нагрузка и т.д. Это позволяет посетителям отслеживать свой прогресс и улучшать свои результаты.
2. Удобство: база данных может быть доступна для посетителей через веб-интерфейс, что позволяет им удобно отслеживать свой прогресс и планировать свои тренировки.

В работе будут рассмотрены следующие задачи:

* Анализ требований к базе данных для фитнес-клуба;
* Проектирование структуры базы данных;
* Создание таблиц и связей между ними;
* Наполнение базы данных тестовыми данными;
* Создание запросов для извлечения информации из базы данных;
* Разработка интерфейса для работы с базой данных.

Для решения поставленных задач будет использоваться язык SQL и СУБД PostgreSQL. Также будет реализован сервер на языке JavaScript с помощью Node.js и фреймворка Express. Результатом работы станет функционирующая база данных для фитнес-клуба, которая будет обеспечивать учет тренировок пользователей и хранение статистики их показателей тела.

# Теоретические сведения

## Диаграмма Ганта

Диаграмма Ганта (Gantt chart) – это один из инструментов управления проектами, иллюстрирующий календарный план проекта и ход его выполнения. Она предоставляет изображение процесса работы в виде графа, что помогает определить приоритеты и последовательность выполнения текущих задач в проекте. Благодаря наглядному представлению временных зависимостей становится возможным более точно рассчитать сроки завершения проекта, определить оптимальное время для запуска разработки и предсказать вероятные задержки.

Временная шкала отображается на горизонтальной оси, а список задач отображается на вертикальной оси. В большинстве случаев она состоит из трех частей: в левой части приведена таблица заданий, в верхней — временная шкала, и в основной части – сама диаграмма зависимости порядка выполнения заданий от времени. Каждая задача имеет свои даты начала и завершения, зависимости ее начала, то есть возможности приступить к ней, от окончания других задач. Вершинами графа являются блоки задач (событий), а ребрами – их зависимости друг от друга. Диаграмма Ганта дополнительно может включать в себя вехи, контрольные точки и исполнителей.

Компонент временной шкалы должен:

* предоставлять пользователю навигацию во временном пространстве;
* отображать текущий временной интервал (в поле видимости) и разделять его на временные промежутки (в большинстве случаев равные), например, день или месяц;
* ппп
* ррр
* поддерживать прокрутку горизонтальным колесом мыши или зажатием левой кнопкой мыши (перетягиванием), сдвигая влево/вправо интервал по временной шкале, не меняя его ширины;
* поддерживать изменение масштаба прокруткой средней кнопкой мыши, увеличивая размер делений/промежутков, относительно текущего положения курсора;
* при достижении определенной ширины, которая задается пользователем, переходить на следующий временной уровень;

Компонент диаграммы должен:

* отображать блоки задач (событий) в соответствующих временных точках;
* поддерживать масштабирование в соответствии с временной шкалой;

## Инструменты web-разработки

HTML – Hyper Text Markup Language – Язык разметки гипертекста. это код, который используется для структурирования и отображения веб-страницы и её контента. технологии разработки сайтов страница, которую понимает веб-браузер.

Кроме HTML-кода загружаются на компьютер пользователя и обрабатываются на нем CSS - каскадные таблицы стилей, и программы (обычно называемые скриптами) на таких языках, как Java, JavaScript, TypeScript.

CSS – это инструмент, позволяющий придать единый вид различным типовым элементам сайта, в частности - легко настраивать и изменять облик всех элементов страницы.

События – это действия или случаи, возникающие в программируемой системе, о которых система сообщает для дальнейшей возможности с ней взаимодействовать. При возникновении события система генерирует сигнал, а также предоставляет механизм, с помощью которого можно автоматически предпринимать какие-либо действия (например, выполнить определённый код), когда происходит событие. В Web события запускаются внутри окна браузера и, как правило, прикрепляются к конкретному элементу, который в нем находится. Это может быть один элемент, набор элементов, документ HTML, загруженный на текущей вкладке, или все окно браузера. Каждое доступное событие имеет обработчик событий – блок кода (обычно это функция JavaScript), который будет запускаться при срабатывании события.

Примеры обработчиков событий:

* elem.onclick – действие при нажатии кнопкой мыши на элемент
* elem.onmouseover – действие при наведении курсора на элемент

## Векторная графика

На сегодняшний день широко распространены два вида компьютерной графики: векторная графика и растровая графика. Каждая из них имеет свои преимущества и недостатки.

Растровый метод предоставляет изображение в виде прямоугольной матрицы, каждая ячейка которой представлена цветной точкой (пикселем). Такие изображения обеспечивают максимальную реалистичность, так как в цифровую форму переводится каждый фрагмент оригинала. В файлах сохраняется информация о цвете каждого пикселя, что влияет на размер самого файла при его сохранении.

К достоинствам растровой графики относятся ее аппаратная реализуемость, программная независимость (форматы файлов для сохранения изображений являются стандартными), а также фотореалистичность полученных изображений. Главными недостатками этого метода являются размер сохраняемого файла (для 8-битной кодировки цвета, каждый пиксель имеет размер одного байта), сложность трансформирования таких изображений с сохранением их качества, а также вытекающее из этого ухудшение качества при масштабировании.

Векторная графика – это вид графической представления, который использует векторы для представления изображения. Векторная графика используется для создания изображений высокого качества, которые могут быть масштабированы без потерь качества. Это позволяет дизайнерам и разработчикам создавать интерактивные и профессиональные изображения и анимации.

Векторный метод описывает изображение с помощью графических примитивов, которые рассчитываются по конкретным математическим формулам. В конечном итоге формируется изображение в виде совокупности отрезков и дуг, которые характеризуют тот или иной объект. Сами изображения состоят из контуров, которые в свою очередь состоят из нескольких смежных сегментов, ограниченных узлами. Так как контур является строго математическим понятием, то он не имеет толщины. Чтобы сделать его видимым ему создают обводку заданных толщины и цвета, проведенных строго по самому контуру. Замкнутые контуры могут иметь внутреннюю заливку.

В отличие от растровой графики, векторный метод предоставляет полную свободу трансформации, дает более высокую точность изображений и занимает значительно меньший объем памяти. К недостаткам относится невозможность применения множества эффектов, используемых при работе с растровыми изображениями, а также невозможность перевести растровое изображение в векторное.

Для построения графов больше всего подходит векторная графика.

Масштабируемая векторная графика (**SVG**) – это формат двумерного векторного изображения на основе синтаксиса [XML](https://developer.mozilla.org/ru/docs/Glossary/XML).

[W3C](https://developer.mozilla.org/ru/docs/Glossary/W3C) начал работу над SVG в конце 1990-х годов. Все основные [браузеры](https://developer.mozilla.org/ru/docs/Glossary/Browser) теперь поддерживают SVG. HTML5 теперь позволяет прямое встраивание [тегов](https://developer.mozilla.org/ru/docs/Glossary/Tag) SVG в [HTML](https://developer.mozilla.org/ru/docs/Glossary/HTML) документ [2].

Графика создается в браузере, что снижает нагрузку на сервер и время отклика сети, обычно связанные с веб-изображениями. То есть обычно небольшое шаблонное описание отправляется с сервера клиенту. Затем клиент реконструирует изображение на основе полученных формул. Конечный пользователь может взаимодействовать с графикой и изменять ее без необходимости сложной и дорогостоящей связи клиент-сервер.

На основе синтаксиса [XML](https://developer.mozilla.org/ru/docs/Glossary/XML), SVG можно стилизовать с помощью [CSS](https://developer.mozilla.org/ru/docs/Glossary/CSS) и сделать интерактивным с использованием [JavaScript](https://developer.mozilla.org/ru/docs/Glossary/JavaScript).

Общие атрибуты тегов SVG-элементов: stroke – цвет линии, stroke-width – толщина линии, fill – цвет заливки (none – без заливки), а также style – css-стиль элемента.

При построении дуг связей между задачами в диаграмме Ганта будет использован тег path – сложная траектория, которая является самым универсальным из SVG-элементов. Позволяет создавать произвольные фигуры. Форма фигуры задается атрибутом d, значение которого – это набор специальных команд. Эти команды могут быть и в верхнем, и в нижнем регистре. Верхний регистр указывает на то, что применяется абсолютное позиционирование, а нижний – относительное.

Команды, определяющие траекторию и направление фигурной линии:

* M, m – Начальная точка – “Mx,y”
* L, l – Отрезок прямой – “Lx,y”
* C, c – Кубическая кривая Безье – “Cx1,y1 x2,y2 x,y”

x1, y1 – координаты первой контрольной точки;

x2, y2 – координаты второй контрольной точки;

x, y – координаты конечной точки кривой.

* Q, q – Квадратичная кривая Безье – “Qx1,y1 x,y”

x1, y1 – координаты контрольной точки;

x, y – координаты конечной точки кривой.

## Кривая Безье

Кривые Безье используются в компьютерной графике для рисования плавных изгибов. Кривая Безье задаётся опорными точками. Как видно по рисунку 1, их может быть от 2х и больше.

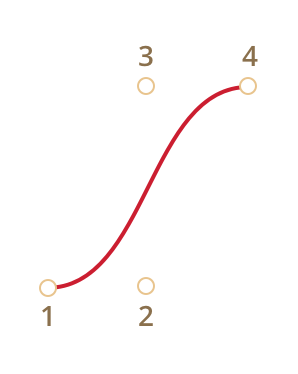
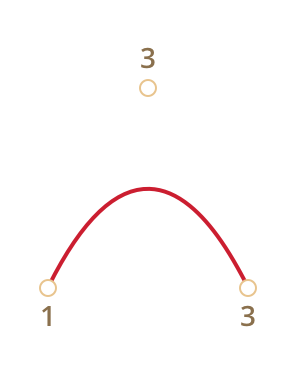
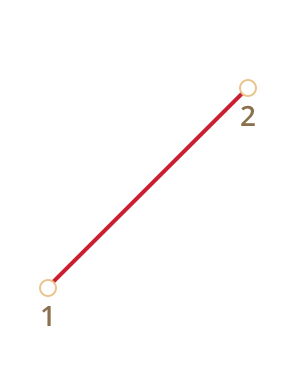


Рисунок 1 – Кривая Безье по 2м, 3м и 4м точкам соответственно [3]

Для двух точек – это линейная кривая (т.е. прямая), для трёх точек – квадратическая кривая (парабола), для четырёх – кубическая. Кривая всегда находится внутри выпуклой оболочки, образованной опорными точками.

* Линейные кривые

Параметр  в функции, описывающей линейный случай кривой Безье, определяет, где именно на расстоянии от  до  находится . Например, при  значение функции соответствует четверти расстояния между точками  и . Параметр  изменяется от до , а  описывает отрезок прямой между точками  и .

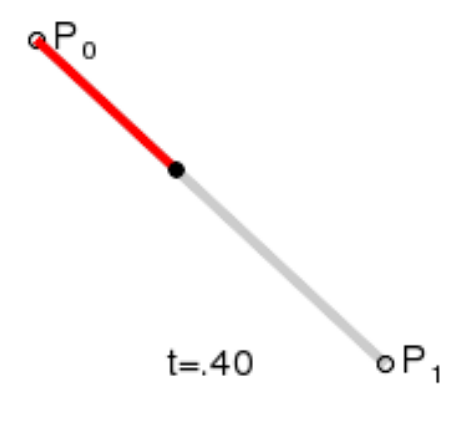


Рисунок 2 – Пример построения линейной кривой [4]

* Квадратичные кривые

Для построения квадратичных кривых Безье требуется выделение двух промежуточных точек  и  из условия, чтобы параметр  изменялся от до :

* Точка  изменяется от  до  и описывает линейную кривую Безье.
* Точка  изменяется от  до  и также описывает линейную кривую Безье.
* Точка  изменяется от  до и описывает квадратичную кривую Безье.

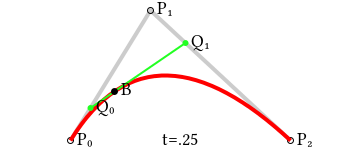


Рисунок 3 – Построение квадратичной кривой Безье [4]

* Кривые высших степеней

Для построения кривых высших порядков соответственно требуется больше промежуточных точек. Для кубической кривой это промежуточные точки ,  и , описывающие линейные кривые, а также точки  и , которые описывают квадратичные кривые: более простое уравнение�0�0�0�1=�1�1�1�2=��0�1�0.

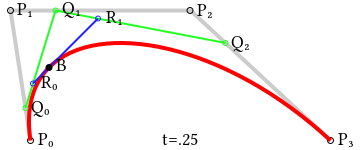


Рисунок 4 – Построение кубической кривой Безье [4]

# Реализация

В данной части содержится описание реализации приложения.

Проект разработан с использованием языка разметки HTML, языка каскадных таблиц стилей CSS и языка программирования TypeScript. Репозиторий проекта находится на странице [1].

## Реализация временной шкалы

Шкала должна поддерживать прокрутку, масштабирование и смену временных уровней, что показано на рисунках 6, 7, и 8 соответственно.

Шкала разделена на временные интервалы (блоки). Каждой координате соответствует свое время (в миллисекундах) и наоборот. Это осуществляется благодаря тому, что известны текущая дата первого блока, проходящего нулевую координату, величины сдвига первого блока относительно нулевой координаты и т.н. «плотности» (density) - количестве миллисекунд в 1 пикселе. Эти данные изменяются при прокрутке, масштабировании и смене временного уровня. То есть, имея значение координаты, можно получить соответствующее ей время (в миллисекундах, начиная с 1.01.1970).

При инициализации приложения, создается экземпляр объекта Date [10], фиксирующий текущее время и часовой пояс пользователя. Текущая плотность рассчитывается из заданных пользователем (или по умолчанию) начальной ширины ячейки и количества миллисекунд в одном временном интервале.

Пример:

Пусть первый временной уровень – день;

Начальная ширина ячейки – .

В дне .

Тогда плотность будет равна .

Теперь необходимо вычислить разницу между текущим временем и временем начала дня. На рисунке 5 показан пример как выглядит первая ячейка в момент времени запуска 20.02.2023 07:12. Сдвиг ячейки равен ;

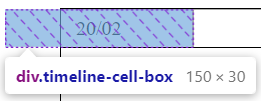




Рисунок 5 – Установка текущего времени в нулевую координату

Начало временного интервала (например, начало дня - в 00:00:00 по местному времени) рассчитывается для каждого уровня агрегации по-своему: для одинаковых временных промежутков, как дни или недели, достаточно применить формулу (1):

,  *(1)*

где

– текущее время,

– начальное время временного промежутка, которому принадлежит время ,  
 – кол-во миллисекунд в одном временного промежутке (например, в дне )

Для месяцев необходимо создать новый экземпляр объекта Date, указав уже известные год, месяц и 1 число (время автоматически будет установлено 00:00:00), получить время в миллисекундах.

Координату сдвига шкалы получаем по формуле (2):

, *(2)*

где

– текущее время в нулевой координате [мс],

– начальное время текущего временного промежутка [мс],

– плотность [мс/пкс]

Для отрисовки шкалы, необходимо знать ширину каждой ячейки, у дней и недель она постоянная, у месяцев, кварталов, лет зависит от количества дней в них. Ширина ячейки зависит от количества миллисекунд во временном интервале и от текущей плотности.

Временные уровни описывает интерфейс aggregationLevel, для всех уровней создается объект со своими свойствами и обработками событий.

К каждому уровню применяется своя итерация в методе setDate, принимающим объект Date и кол-во итераций. Например, для недель необходимо получить номер дня в месяце, прибавить к нему количество итераций, помноженное на семь и к переданному объекту Date применить метод setDate, меняющий число месяца. Если метод принял в качестве параметра отрицательное число или число, превышающее количество дней в месяце, то месяц сменяется. Для итерации по месяцам применяется метод setMonth.

Перемещение по шкале реализует метод shift. При прокрутке обновляются дата первого блока и координата сдвига. Величина сдвига колеблется от 0 до ширины ячейки. При проходе границы временного интервала, добавляется новая ячейка с одной стороны и удаляется с другой в зависимости от направления прокрутки; значение сдвига сменяется с 0 на ширину ячейки и наоборот.

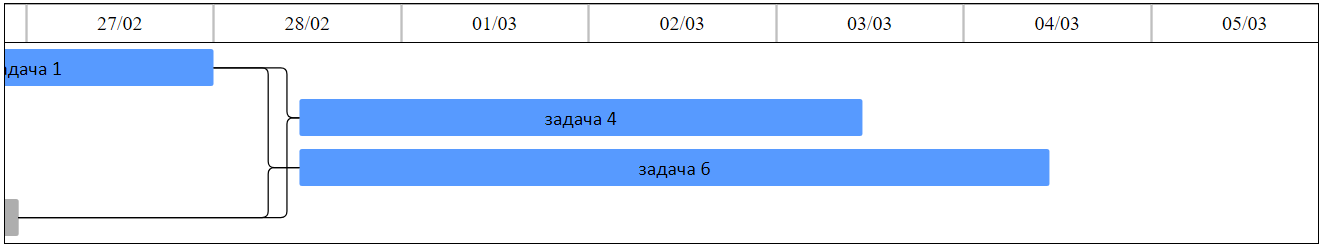
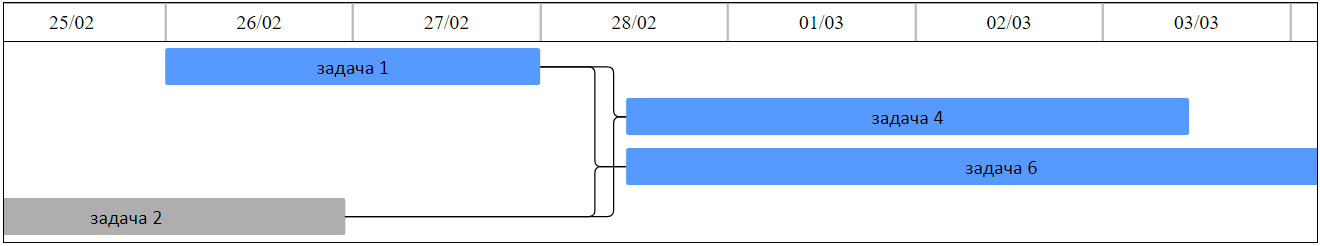
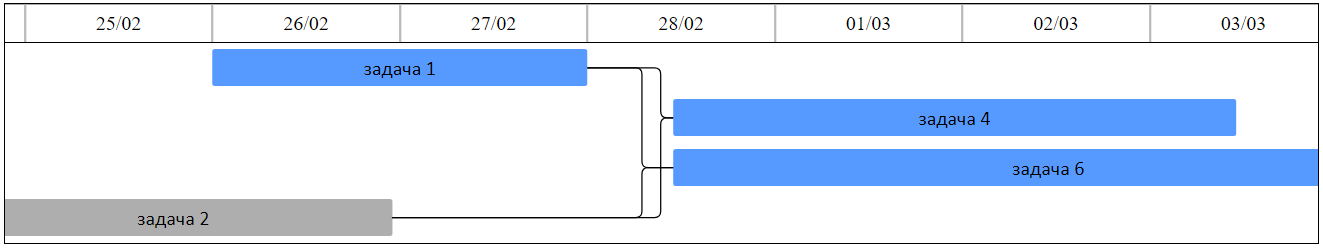


Рисунок 6 – Пример работы прокрутки

Масштабирование шкалы реализует метод scale. При масштабировании обновляются дата первого блока, координата сдвига и плотность, а также количество ячеек. Масштабирование происходит относительно курсора. Расстояние от начала шкалы до положения курсора умножается на изменение масштаба и обновляется, для чего применяется вышеописанная прокрутка. Недостающие ячейки дорисовываются, лишние удаляются.



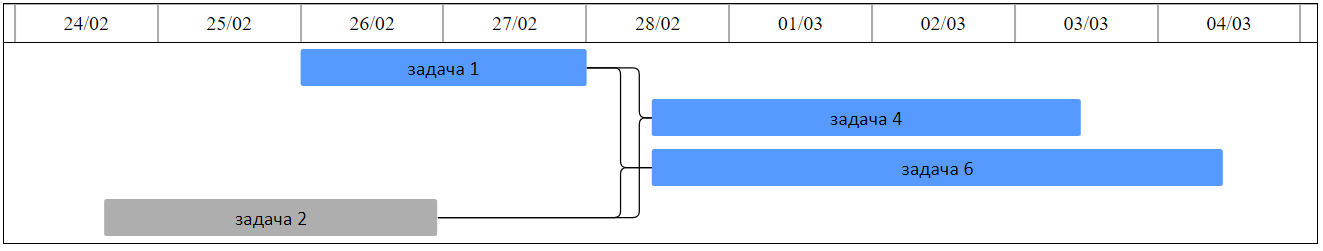
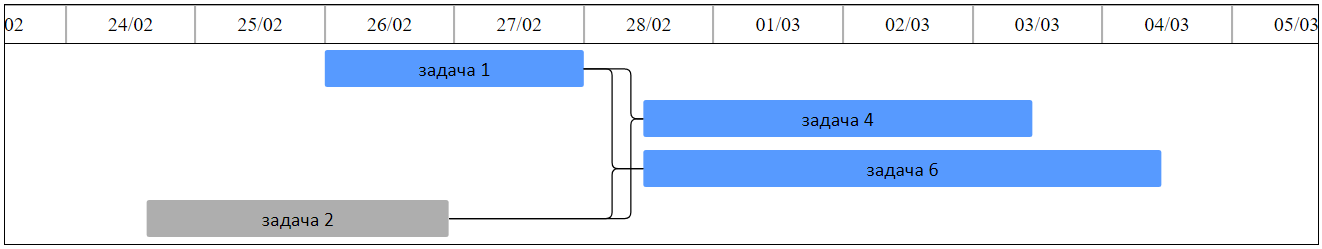


Рисунок 7 – Пример масштабирования (курсор поставлен на границу между блоками “27/02” и “28/02”)

Если достигается определенная ширина, то сменяется временной уровень. Уровни задаются пользователем массивом или по умолчанию: day – week – month. При смене идет продвижение по массиву. При попытке уменьшить минимальный уровень (либо увеличить максимальный) ничего не произойдет. Когда сменяется уровень, не обновляются плотность и положение временной шкалы, но меняются ширина и количество ячеек, а также максимальная и минимальная ширина ячеек.



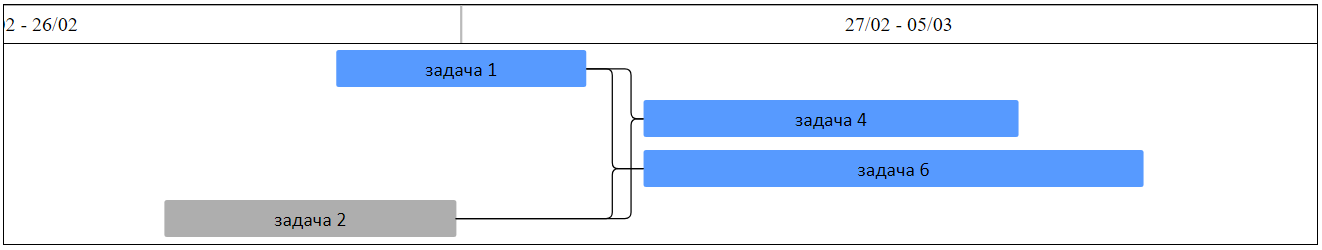


Рисунок 8 – Пример смены временного уровня

## Реализация отображения блоков задач

Визуализация диаграммы выполнена в упрощенном варианте, без приведения списка задач на вертикальной оси, и основывается на исходных данных, где каждой задаче присвоено время начала и время дедлайна, а также свой идентификатор. Для каждого времени высчитывается координата на временной шкале по формуле (3) и событие располагается в соответствии с ними.

, *(3)*

где

– время [мс],

– начальное время блока в нулевой координате [мс],

– плотность [мс/пкс],

– величина сдвига шкалы

Создается элемент div. Координата начала устанавливается в его css-свойство margin-left, а ширина блока width равна разнице между координатами. Если значение поля completed указано как “true”, то блок закрашивается серым цветом, иначе – голубым. За цвет блока отвечает css-свойство background-color. Высота диаграммы зависит от количества полученных событий, поэтому в значение поля height установлено auto. Пример отрисовки представлен на рисунке 9. При наведении на блок задачи на экран выводится подсказка – даты и время ее начала и окончания. За это отвечает атрибут title.

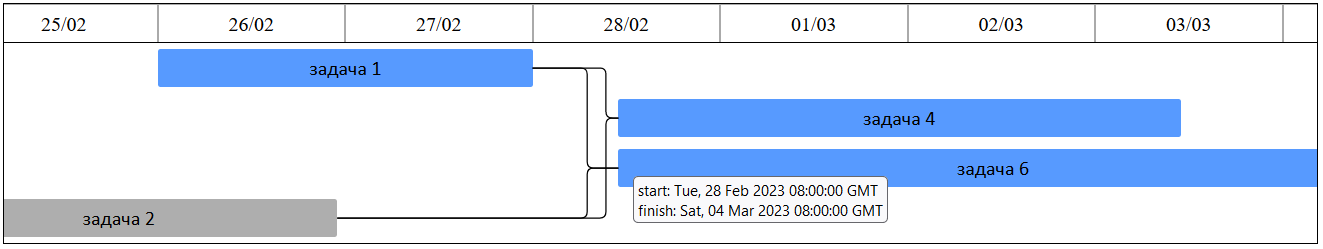


Рисунок 9 – Пример диаграммы Ганта

## Реализация размещения дуг связей между блоками задач

Для отрисовки ребер используется векторная графика. Чтобы вектора отображались на web-странице, создается общий холст, который будет заключать в себе все дуги.

Все связи имеют три поля: идентификатор самой связи, идентификатор задачи-predecessor (задачи, от которой зависит вторая задача) и идентификатор задачи-successor (задачи, зависимой от первой задачи). Если задача-successor начинается раньше, чем заканчивается задача-predecessor, то дуга связи окрашивается в красный цвет, указывающий на нарушение последовательности. Будем называть такую связь неправильной зависимостью. Толщина линии – 1 px, заливка отсутствует.

1. Отрисовка дуг в виде кривых

Для построения дуг используется кубическая кривая Безье. Кривая на рисунке 10 описывается следующим образом: “M, C, ,,”.

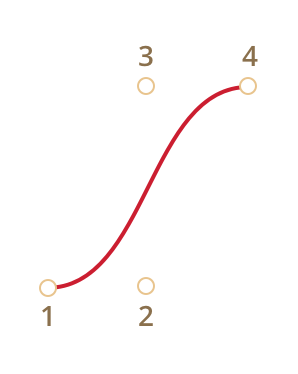


Рисунок 10 – Кубическая кривая Безье

В случае неправильной зависимости схема построения дуги описана на рисунке 11. Координата y точки C считается по формуле (4). Отрезок CD описывается прямой.

(4)

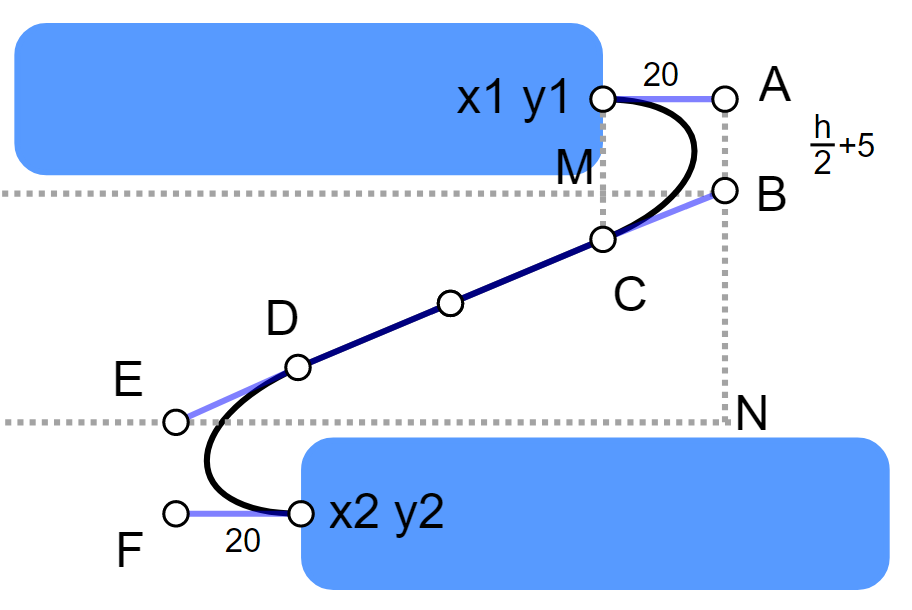


Рисунок 11 – Схема дуги неправильной зависимости [5]

Траектория кривой, показанной на рисунке 11 записывается следующим образом: “M,C LC,”.

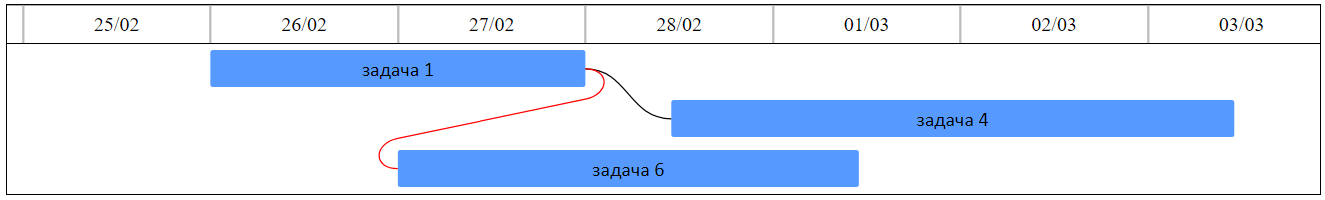


Рисунок 12 – Пример отрисовки дуг в виде кривых

1. Простой алгоритм отрисовки дуг в виде ломанных прямых

При использовании варианта отрисовки простым алгоритмом каждая дуга обрабатывается отдельно. На вход функции подаются две точки – начало (точка завершения задачи predecessor) и конец (точка начала задачи successor), а также высота блока.

Если расстояние по оси X между точками не меньше 5, то фиксируются 4 контрольные точки: начальная, конечная, точка с теми же координатами, что и конечная, но со сдвигом в по оси X, и точка, с той же координатой x, но координата y совпадает с начальной. В результате получаем траекторию “M, , , ,”.

Если расстояние по оси X между точками больше 5, то фиксируются 6 контрольных точек: начальная, конечная, точка со сдвигом от начальной, точка со сдвигом от конечной и две точки либо над, либо под блоком задачи в зависимости от того, ниже или выше находится вторая задача от первой. В результате получаем траекторию “M, , , , , ,”, где – высота блока.

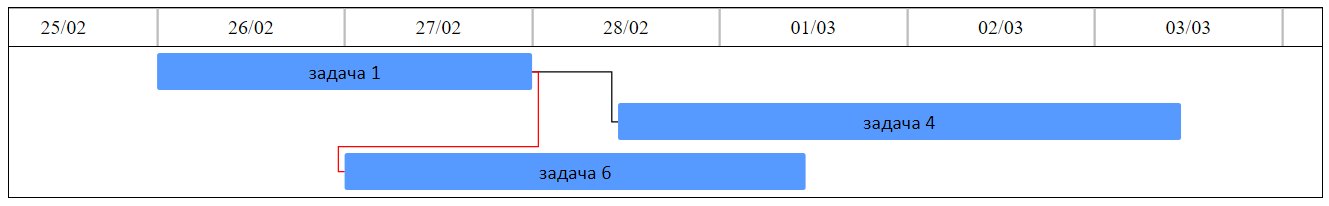


Рисунок 13 – Пример простой отрисовки дуг

1. Сложный алгоритм отрисовки дуг в виде ломанных прямых

На рисунке 14 показан недостаток простого алгоритма – при определенном положении задач ребра накладываются друг на друга и становится невозможным определить взаимосвязь между задачами.

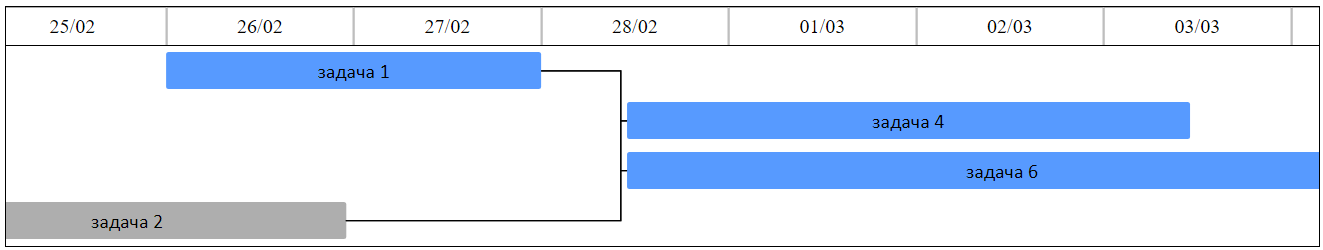


Рисунок 14 – Пример, иллюстрирующий недостаток простого алгоритма

Поэтому было принято решение реализовать размещение дуг, которое бы минимизировало их наложение друг на друга.

Для начала было добавлено округление углов ломанных. По рисунку 15 видно, что стало проще отследить направление поворота кривых.

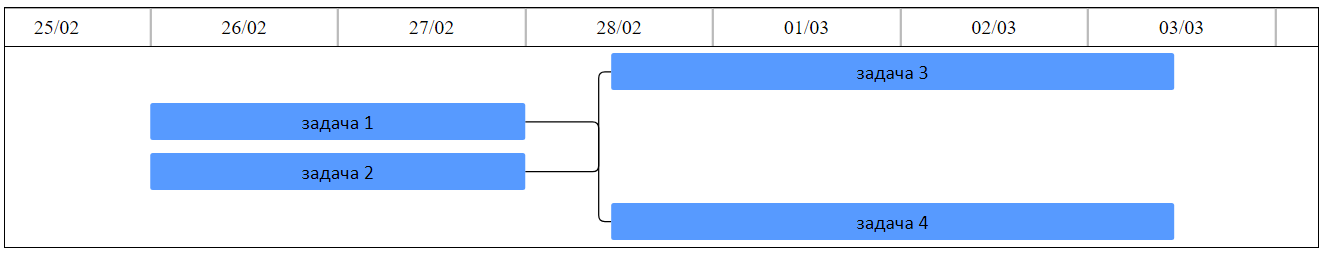


Рисунок 15 – Закругление углов

Для построения закругленных углов использовалась квадратичная кривая Безье. Например, траектория угла, показанного на рисунке 16, описывается следующим образом: “M, Q, ,”.

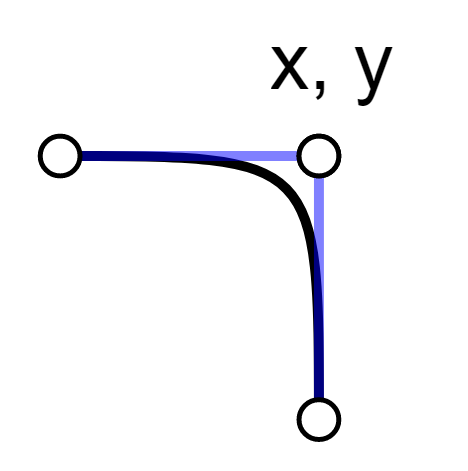


Рисунок 16 – Квадратичная кривая Безье [5]

В отличие от простого алгоритма, на вход функции позиционирования ребер подаются оба массива данных: массив задач и массив зависимостей, так как для расчета размещения дуг важно учитывать положение всех элементов диаграммы.

Первым шагом необходимо отсортировать зависимости начала и конца каждой задачи в порядке их расположения по оси Y. Чтобы избежать пересечений, следует к тому же учитывать особый случай – неправильную зависимость. Она должна быть в начале отсортированных зависимостей, если находится выше исходной задачи, или в конце, если ниже. Если таких зависимостей несколько, то они сортируются между собой.

Далее проходим второй раз по отсортированному массиву. Зависимости делятся на две части – те, что выше исходной задачи, и те, что ниже. Назначаем каждой зависимости ее порядок. Считаем максимальное число связей среди двух частей.

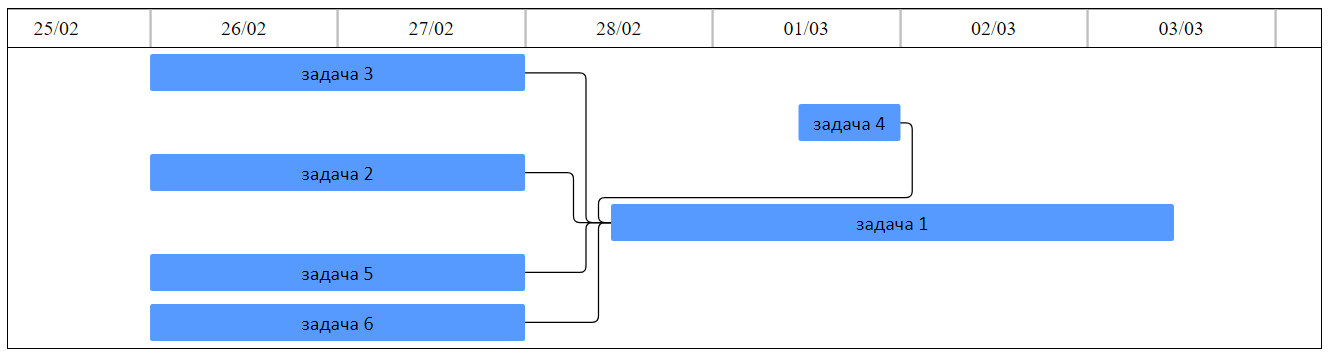


Рисунок 17 – Сортировка зависимостей

Следующим шагом сортируем блоки задач по дате их начала (successors) в обратном порядке. Создаем для каждой новый атрибут – координата отступа, изначально равный координате начала минус 10. Если разница между координатами отступа двух задач меньше 10, то новая координата отступа считается по формуле (5).

*, (5)*

где

– координата отступа задачи,

– число связей предыдущей задачи (на рисунке 17, у “задачи 1” – 3 связи сверху)

Аналогично для даты окончания задач (predecessors), но сортировка должна быть в правильном порядке.

Результат данного алгоритма представлен на рисунке 18.

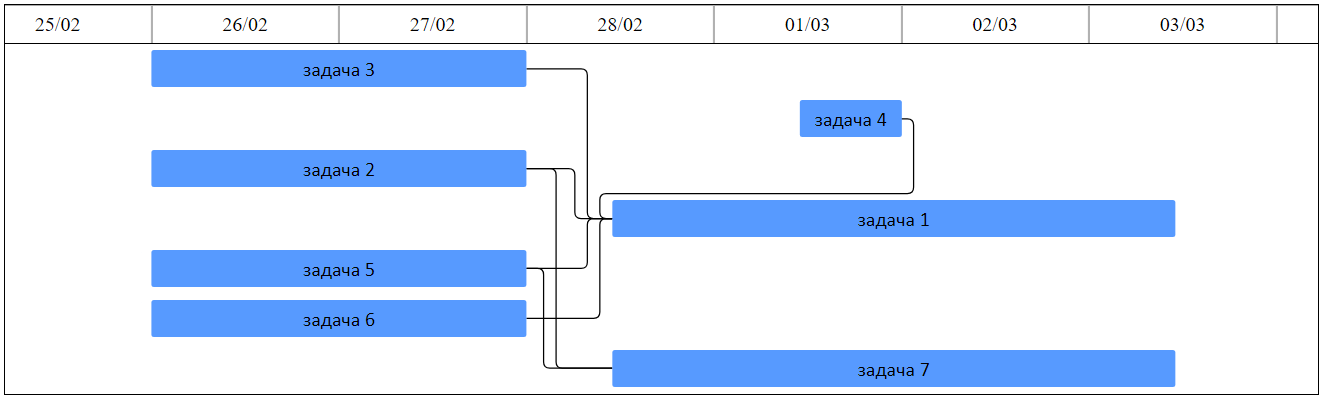


Рисунок 18 – Пример сложной отрисовки дуг в виде ломанных прямых

Более того, можно добавить разделение ребер, привязанных к одной задаче. Для этого необходимо разделить высоту блока задачи на число привязанных зависимостей плюс 1, и посчитать координаты концов дуг в соответствии с их порядком в отсортированном массиве.

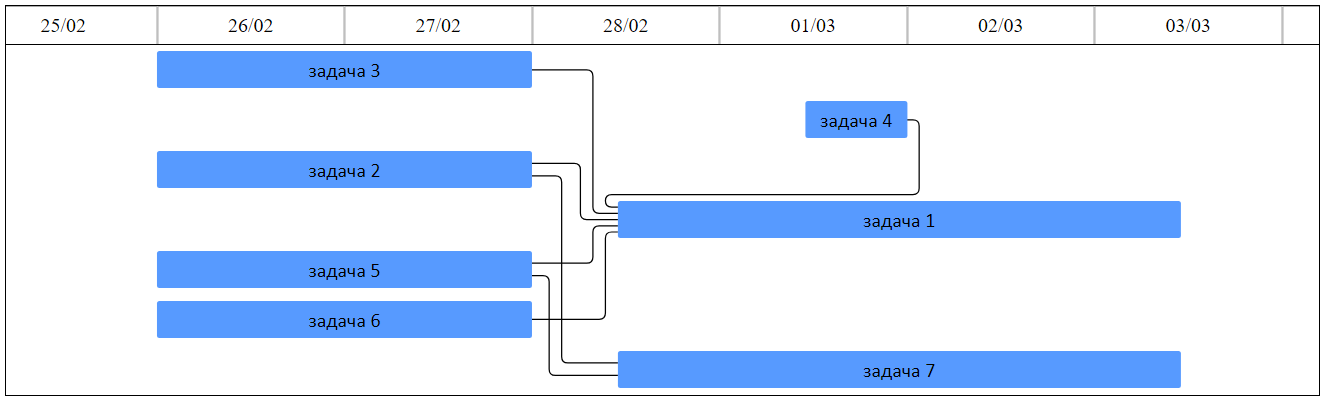


Рисунок 19 – Разделение ребер, привязанных к одной задаче

1. Отсечение дуг

Для того, чтобы найти пересечения ребер, нужно создать две хеш-таблицы, в которые необходимо добавить вертикальные и горизонтальные части каждой ломанной попиксельно. Если по одному ключу в двух хеш-таблицах находятся две точки, то значит в ней есть пересечение. Чтобы добавить отсечение кривой, необходимо в траекторию вставить следующую запись: “, M,”.

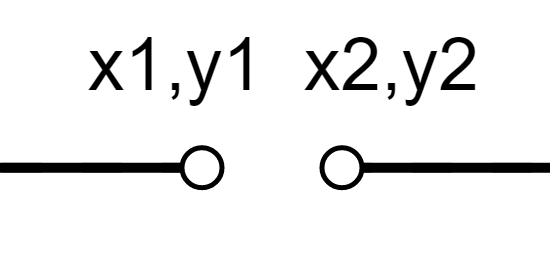


Рисунок 20 – Отсечение кривой [5]

## Функциональные возможности приложения

Интерфейс страницы представлен на рисунке 21.

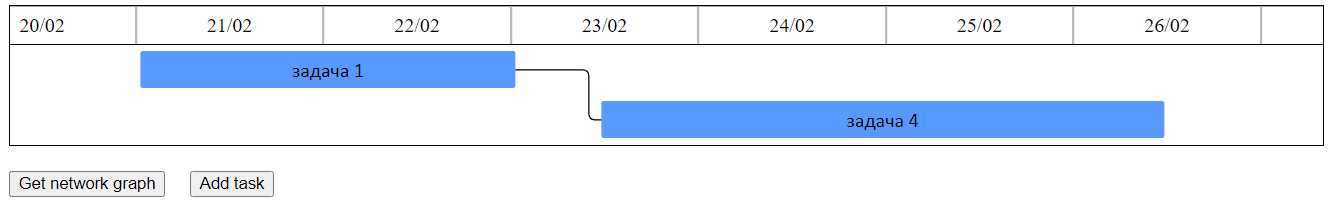


Рисунок 21 – Интерфейс приложения

Помимо визуального представления календарного плана работ, разрабатываемое приложение имеет следующие возможности:

1. Добавление задач.

Для того, чтобы добавить задачу необходимо нажать на кнопку “Add task”. После этого откроется окно, показанное на рисунке 22. Оно содержит в себе 4 поля: имя задачи, даты ее начала и окончания, и флаг, указывающий на то, выполнена ли задача.

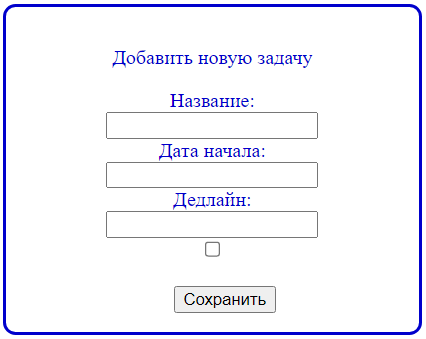


Рисунок 22 – Интерфейс окна добавления новой задачи

Имя задачи представлено строковым типом данных. Поле completed имеет логический тип данных: если поставить галочку на флаг (checkbox), то в его значение будет установлено true, иначе – false.

Формат даты должен соответствовать [стандарту ISO 8601](https://go.microsoft.com/fwlink/?LinkId=156016) [6]:

* YYYY-MM-DD
* YYYY-MM-DDThh:mm<TZDSuffix>
* YYYY-MM-DDThh:mm:ss<TZDSuffix>

Для значения даты в этих форматах YYYY представляет собой четырехзначное представление года, MM двухзначное представление месяца, начинающееся с 1, и DD двухзначное представление дня, начинающееся с 1.

Если включено необязательное значение времени, то для отделения значения времени от значения даты требуется литеральный оценщик T. Для значения hh времени — представление часа в 24-часовой нотации, mm двухзначное представление минуты и ss двухзначное второе представление. Каждое из этих значений выражается в виде целого числа, начиная с нуля.

Литеральное значение Z в конце строки указывает, что время выражается в формате UTC.

Ниже приведен пример объединенного формата UTC.

Значение даты и времени для 20 февраля 2023 г. в 07:12:03 по времени UTC выражается следующим образом:

2023-02-20T07:12:03Z

Пример добавления новой задачи описан на рисунках 23-24.

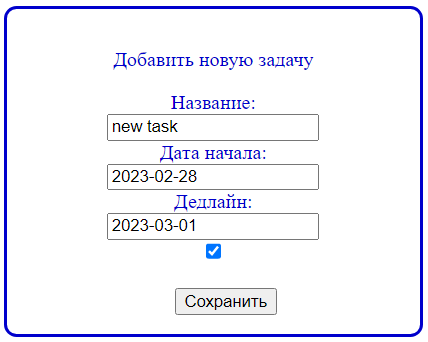


Рисунок 23 – Пример заполнения формы добавления задачи

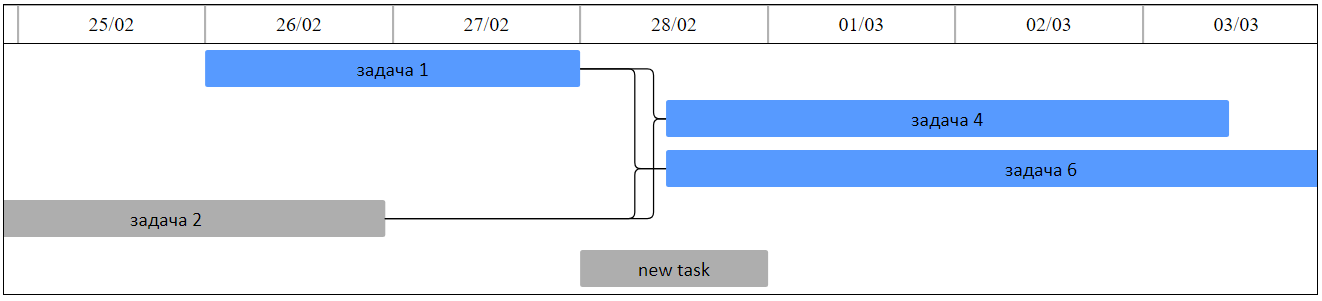
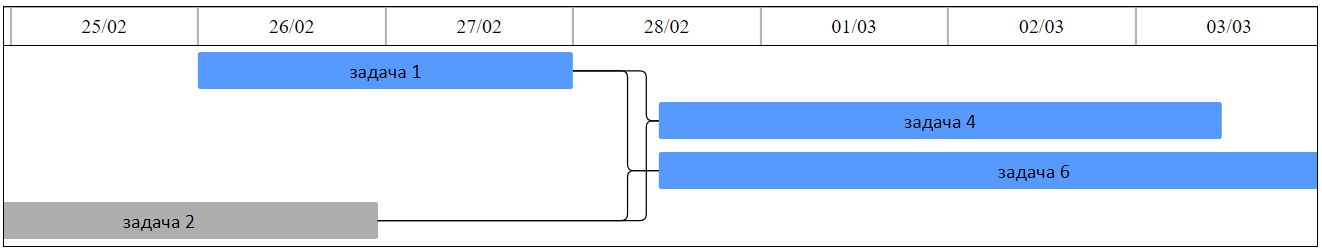


Рисунок 24 – Результат добавления новой задачи

1. Удаление задач

Для того, чтобы удалить задачу, необходимо нажать на нее кнопкой мыши с зажатой клавишей Shift. При этом удаляются все зависимости данной задачи.

На рисунке 25 была удалена “задача 6”.



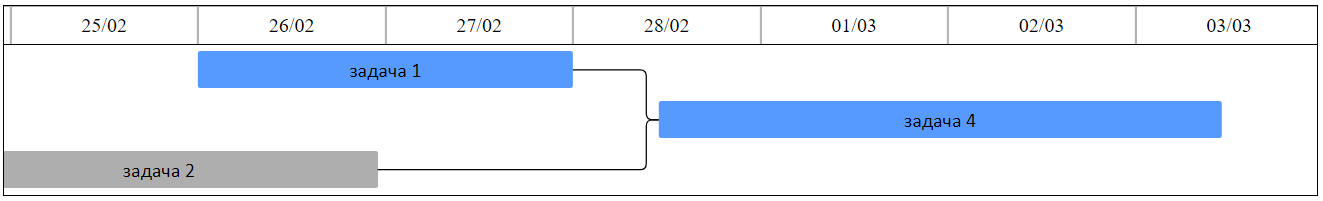
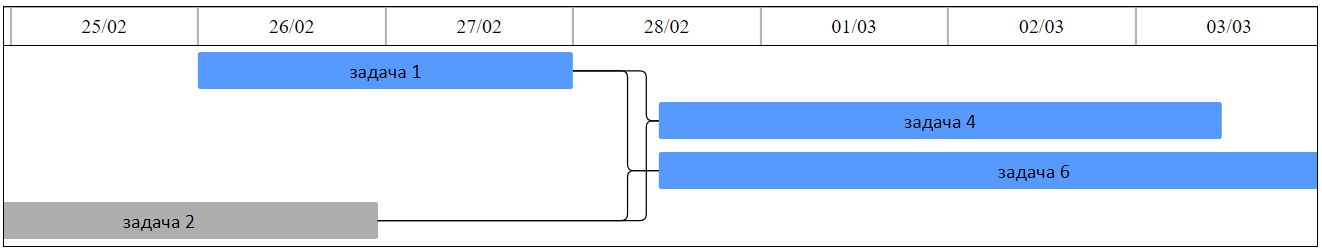


Рисунок 25 – Пример удаления задачи

1. Удаление связей

Для того, чтобы удалить связь, необходимо навести на нее курсор с зажатой клавишей Shift.

На рисунке 26 была удалена связь между блоками “задача 2” и “задача 6”.



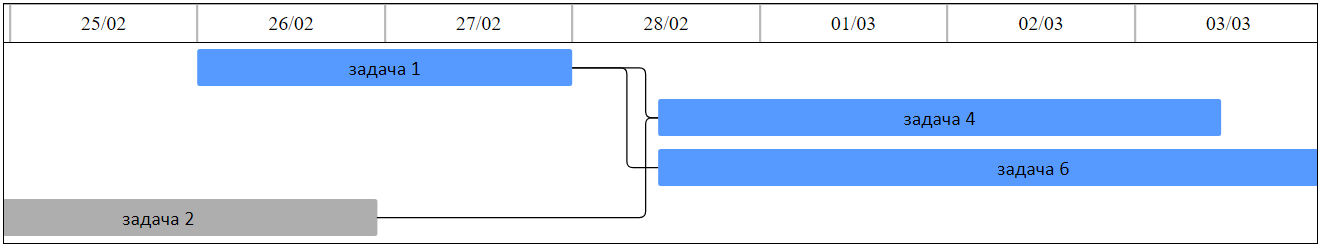


Рисунок 26 – Пример удаления связи

1. Добавление связей

Для того, чтобы добавить связь, необходимо нажать кнопкой мыши на задачу-predecessor (т.е. задачу, от которой зависит вторая) с зажатой клавишей Alt, далее таким же образом назначить задачу-successor (т.е. задачу, зависимую от первой).

На рисунке 27 была добавлена связь между блоками “задача 2” и “задача 6”.

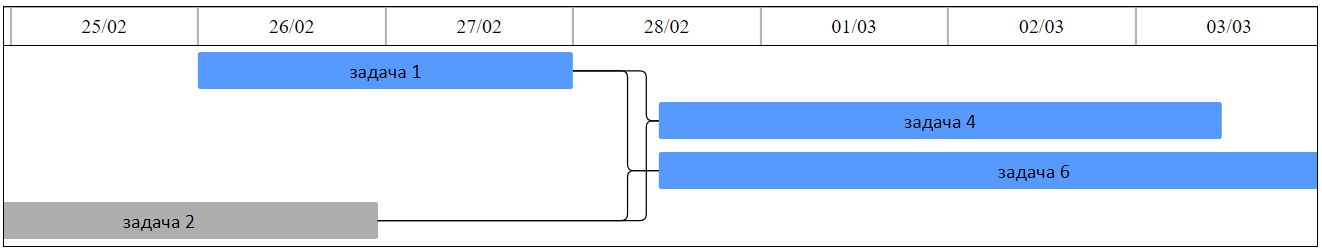
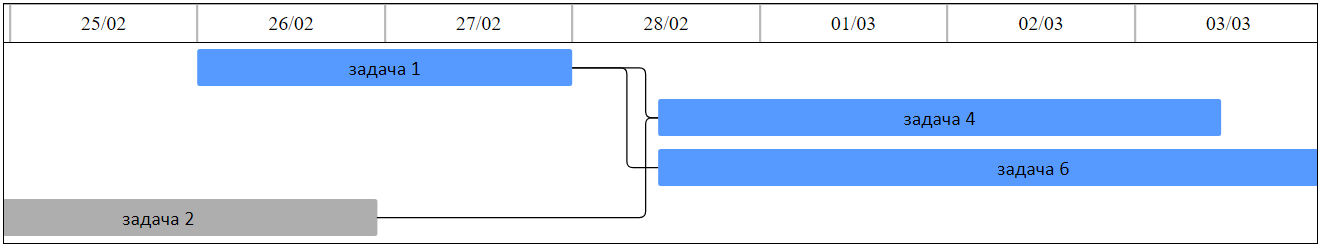
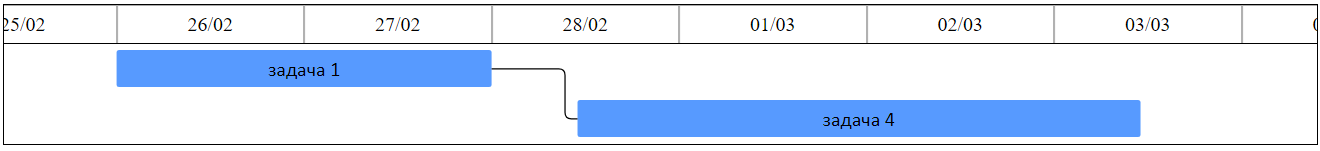


Рисунок 27 – Пример добавления связи

1. Смена флага выполнения задачи

При нажатии кнопкой мыши на блок задачи с зажатой клавишей Ctrl задача сменит свой флаг, указывающий на то, выполнена ли она. На рисунке 28 изначально “задача 4” считалась не завершенной, но после ее цвет стал серым. Это означает, что теперь она выполнена.



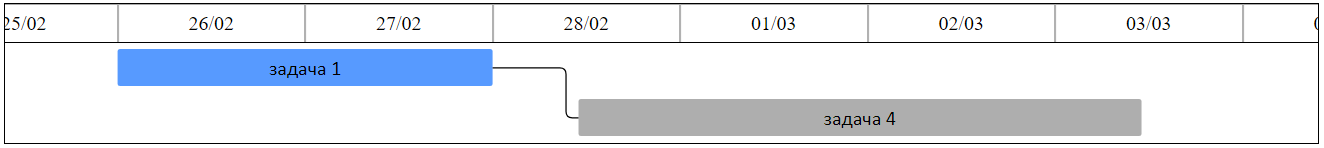


Рисунок 28 – Пример смены флага выполнения задачи

1. Получение представления диаграммы в виде сетевого графика.

На сетевом графике задачи изображаются блоками, соединенными стрелками в соответствии с взаимосвязями работ. В отличие от диаграммы Ганта, сетевой график позволяет визуально определить последовательность и зависимость задач, абстрагируясь от запланированного времени их выполнения.

При нажатии на кнопку “Get network graph” исходные данные преобразовываются в граф, описанный на языке Dot, из которого можно получить изображение, например, средствами Graphviz [7]. Пример полученного представления диаграммы в виде сетевого графика изображен на рисунке 29.

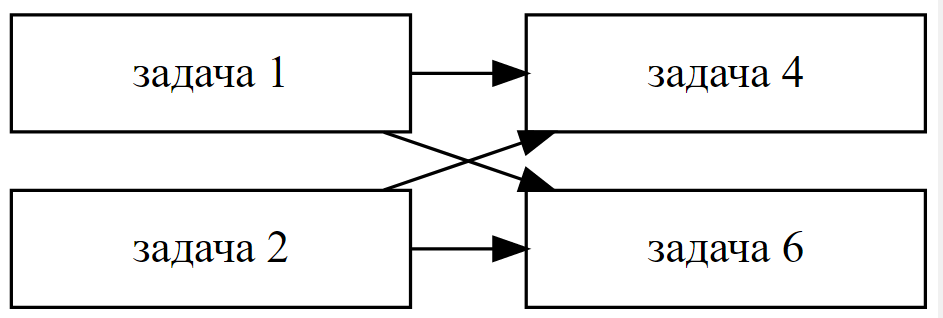
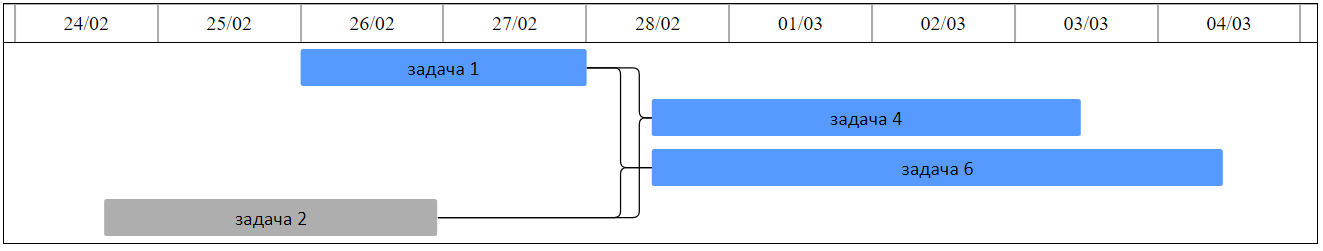


Рисунок 29 – Пример сетевого графика

# Оценка визуализации и обзор аналогов

Как можно заметить по рисунку 30, в приведенных готовых реализациях диаграммы Ганта дуги зависимостей могут накладываться друг на друга при определенном положении блоков задач. Однако в рамкой данной работы удалось реализовать возможность размещения ребер таким образом, чтобы, глядя на каждую из связей, можно было определить к чему она привязана.

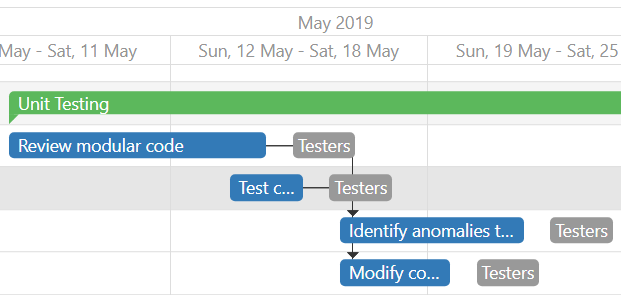


Рисунок 30 – диаграмма Ганта (а) – OnlyOffice [8], (б) – DevExtreme [9]

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы были рассмотрены различные виды представления графика работ бизнес-проекта, такие как диаграмма Ганта и сетевой график, и технологии их визуализации средствами компьютерной графики. На основе изученной информации была разработана программа, позволяющая демонстрировать и редактировать план проекта.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Репозиторий проекта – <https://github.com/mathhyyn/gantt>
2. Документация по svg-графике – <https://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html>
3. Иллюстрации кривой Безье – <https://learn.javascript.ru/bezier-curve>
4. Иллюстрации кривой Безье – https://ru.wikipedia.org/wiki/Кривая\_Безье
5. Иллюстрации получены с помощью векторного редактора – https://bmstu-iu9.github.io/ptp2021-3-vector-editor/
6. Стандарт ISO 8601 – <https://www.w3.org/TR/NOTE-datetime>
7. Документация библиотеки Graphviz – [Электронный ресурс] <http://www.graphviz.org/documentation/>
8. Диаграмма Ганта OnlyOffice – https://helpcenter.onlyoffice.com/ru/userguides/groups-guides-projects-gantt-chart.aspx
9. The DevExtreme JavaScript Gantt component – <https://js.devexpress.com/Demos/WidgetsGallery/Demo/Gantt/Overview/jQuery/Light/>
10. JavaScript Date object – https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Date

ПРИЛОЖЕНИЕ

export const tasks = JSON.parse(

    '[{"id": "t1", "title": "задача 1", "completed": false, "date": "2023-02-26T00:00:00.000Z", "deadline": "2023-02-28T00:00:00.000Z"},\

    {"id": "t2", "title": "задача 2", "completed": true, "date": "2023-02-24T15:00:00.000Z", "deadline": "2023-02-26T23:00:00.000Z"}]');

export const dependencies = JSON.parse('[{"id":"d1","predecessor":"t1","successor":"t2"}]');

Листинг A – входные данные: массивы задач и зависимостей

export const drawPath: ArrowDrawer = function (t1: Point, t2: Point, height: number): SVGPathElement {

    const elem = document.createElementNS('http://www.w3.org/2000/svg', 'path');

    let path = 'M ' + t1.x + ',' + t1.y + ' C';

    if (t2.x - t1.x < 0) {

        let Ax = t1.x + 20;

        let Ay = t1.y;

        let Bx = t2.x - 20;

        let By = t2.y;

        let dy = 5; // marginVertical

        if (t1.y <= t2.y) {

            Ay += height / 2 + dy;

            By -= height / 2 + dy;

        }

        else {

            Ay -= height / 2 + dy;

            By += height / 2 + dy;

        }

        let dy1 = (By - Ay) \* (Ax - t1.x) / (Ax - Bx);

        let Cy = Ay + dy1;

        let Dy = By - dy1;

        path += Ax + ',' + t1.y + ' ' + Ax + ',' + Ay + ' ' + t1.x + ',' + Cy;

        path += ' L' + t2.x + ',' + Dy;

        path += ' C' + Bx + ',' + By + ' ' + Bx + ',' + t2.y + ' ' + t2.x + ',' + t2.y;

    }

    else {

        let x22 = (t1.x + t2.x) / 2;

        path += x22 + ',' + t1.y;

        path += ' ' + x22 + ',' + t2.y + ' ' + t2.x + ',' + t2.y;

    }

    elem.setAttribute('d', path);

    if (t2.x < t1.x) elem.setAttribute('stroke', 'red');

    else elem.setAttribute('stroke', 'black');

    elem.setAttribute('fill', 'none');

    return elem;

};

Листинг B – функция отрисовки кривых

export const drawPolyline: ArrowDrawer = function (t1: Point, t2: Point, height: number): SVGPathElement {

    const elem = document.createElementNS('http://www.w3.org/2000/svg', 'path');

    let path = 'M ' + t1.x + ',' + t1.y + ' ';

    let x22 = t2.x - 5;

    if (x22 - t1.x < 5) {

        let x11 = t1.x + 5;

        let y22 = t2.y;

        if (t1.y <= t2.y) y22 -= height / 2 + 5;

        else y22 += height / 2 + 5;

        path += x11 + ',' + t1.y + ' ' + x11 + ',' + y22 + ' ' + x22 + ',' + y22;

    }

    else {

        path += (x22 - 5) + ',' + t1.y + ' Q ' + x22 + ',' + t1.y + ' ' + x22 + ',';

        if (t2.y > t1.y) path += (t1.y + 5);

        else path += (t1.y - 5);

        path += 'L';

    }

    if (t2.y > t1.y) path += ' ' + x22 + ',' + (t2.y - 5);

    else path += ' ' + x22 + ',' + (t2.y + 5);

    path += ' Q ' + x22 + ',' + t2.y + ' ' + t2.x + ',' + t2.y;

    elem.setAttribute('d', path);

    if (t2.x < t1.x) elem.setAttribute('stroke', 'red');

    else elem.setAttribute('stroke', 'black');

    elem.setAttribute('fill', 'none');

    return elem;

};

Листинг C – функция отрисовки ломанных простым алгоритмом

function assignDependenciesToEvents(arrows: {}, events: Map<string, Event>) {

    for (let ar in arrows) {

        let arrow = arrows[ar];

        events[arrow.successor].startDependencies.push(arrow);

        events[arrow.predecessor].endDependencies.push(arrow);

    };

}

function sortSuccessors(events: Map<string, Event>): Event[] {

    let result: Event[] = [];

    for (let ev in events) {

        let event = events[ev];

        if (event.startDependencies.length > 0)

            result.push(event);

    };

    result.sort((a: Event, b: Event) => { return a.x1 - b.x1; });

    //result.reverse();

    return result;

}

function sortPredecessors(events: Map<string, Event>): Event[] {

    let result: Event[] = [];

    for (let ev in events) {

        let event = events[ev];

        if (event.endDependencies.length > 0)

            result.push(event);

    };

    result.sort((a: Event, b: Event) => { return a.x2 - b.x2; });

    //result.reverse();

    return result;

}

function sortDependencies(events: Map<string, Event>, height: number) {

    for (let ev in events) {

        events[ev].startDependencies.sort((a: Arrow, b: Arrow) => {

            if (a.t2.x - a.t1.x < 0 && b.t2.x - b.t1.x >= 0)

                if (a.t1.y < a.t2.y) return -1;

                else return 1;

            if (b.t2.x - b.t1.x < 0 && a.t2.x - a.t1.x >= 0)

                if (b.t1.y < b.t2.y) return 1;

                else return -1;

            return a.t1.y - b.t1.y;

        });

        events[ev].endDependencies.sort((a: Arrow, b: Arrow) => {

            if (a.t2.x < a.t1.x && b.t2.x < b.t1.x && a.t2.y >= a.t1.y && b.t2.y >= b.t1.y)

                return b.t2.y - a.t2.y;

            return a.t2.y - b.t2.y;

        });

    }

    for (let ev in events) {

        let event = events[ev];

        let i = 0;

        let j = 0;

        event.leftMargin = 0;

        let dependenciesIsFromBelow = false;

        for (let ar of event.startDependencies) {

            if (ar.t1.y >= ar.t2.y && !dependenciesIsFromBelow) {

                event.leftMargin = Math.max(i, event.startDependencies.length - i);

                i = event.startDependencies.length - i - 1;

                dependenciesIsFromBelow = true;

            }

            ar.orderAtSuccessor = i;

            if (ar.t1.y >= ar.t2.y) i--;

            else i++;

            j++;

            //ar.t2.y += j \* height / (event.startDependencies.length + 1);

            ar.t2.y += height / 2

        }

        i = 0;

        j = 0;

        event.rightMargin = 0;

        dependenciesIsFromBelow = false;

        for (let ar of event.endDependencies) {

            if (ar.t2.y >= ar.t1.y && !dependenciesIsFromBelow) {

                event.rightMargin = Math.max(i, event.endDependencies.length - i);

                i = event.endDependencies.length - i - 1;

                dependenciesIsFromBelow = true;

            }

            ar.orderAtPredecessor = i;

            if (ar.t2.y >= ar.t1.y) i--;

            else i++;

            j++;

            //ar.t1.y += j \* height / (event.endDependencies.length + 1);

            ar.t1.y += height / 2

        }

    }

}

export const drawPositionedArrows = function (arrows: {}, events: Map<string, Event>, height?: number): SVGPathElement[] {

assignDependenciesToEvents(arrows, events);

    let successors = sortSuccessors(events);

    let predecessors = sortPredecessors(events);

    sortDependencies(events, height);

    for (let i = 1; i < successors.length; i++) {

        if (events[successors[i].id].x1 - events[successors[i - 1].id].x1 < 10) {

            events[successors[i].id].x1 = events[successors[i - 1].id].x1 - events[successors[i - 1].id].leftMargin \* 10 - 5;

        }

    }

    for (let i = 1; i < predecessors.length; i++) {

        if (events[predecessors[i].id].x2 - events[predecessors[i - 1].id].x2 < 10) {

            events[predecessors[i].id].x2 = events[predecessors[i - 1].id].x2 - events[predecessors[i - 1].id].rightMargin \* 10 - 5;

        }

    }

    let result: SVGPathElement[] = [];

    for (let ev in events) {

        let i = 0;

        for (let arrow of events[ev].startDependencies) {

            i++;

            let t1 = arrow.t1;

            let t2 = arrow.t2;

            const elem = document.createElementNS('http://www.w3.org/2000/svg', 'path');

            let path = 'M ' + t1.x + ',' + t1.y + ' ';

            let x22 = events[arrow.successor].x1 - arrow.orderAtSuccessor \* 10;

            let y22 = events[arrow.successor].y;

            cornerDirection = 5;

            if (events[arrow.successor].y < events[arrow.predecessor].y) cornerDirection = -5;

            if (t2.x - t1.x >= 20) {

                path += (x22 - 5) + ',' + t1.y + ' Q ' + x22 + ',' + t1.y + ' ' + x22 + ',';

                path += (t1.y + cornerDirection);

            } else {

                let x11 = events[arrow.predecessor].x2 + arrow.orderAtPredecessor \* 10;

                path += (x11 - 5) + ',' + t1.y + ' Q ' + x11 + ',' + t1.y + ' ' + x11 + ',';

                path += (t1.y + cornerDirection);

                path += ' L ' + x11 + ',';

                path += (y22 - cornerDirection);

                path += ' Q ' + x11 + ',' + y22 + ' ' + (x11 - 5) + ',' + y22;

                path += ' L ' + (x22 + 5) + ',' + y22 + ' Q ' + x22 + ',' + y22 + ' ' + x22 + ',';

                if (events[arrow.successor].y == events[arrow.predecessor].y) cornerDirection = -5;

                path += (y22 + cornerDirection);

                addVerticalLine(y22 + cornerDirection, x22, t2.y - cornerDirection, arrow);

            }

            path += ' L ';

            path += x22 + ',' + (t2.y - cornerDirection);

            path += ' Q ' + x22 + ',' + t2.y + ' ' + (x22 + 5) + ',' + t2.y + ' L ' + t2.x + ',' + t2.y;

            elem.setAttribute('d', path);

            //if (t2.x < t1.x) elem.setAttribute('stroke', 'red'); else

            elem.setAttribute('stroke', 'black');

            elem.setAttribute('fill', 'none');

            arrow.cell = elem;

        }

    }

    return result;

};

Листинг D – функция отрисовки ломанных сложным алгоритмом