МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ КАЗАХСТАН

СӘТБАЕВ УНИВЕРСИТЕТІ

Институт кибернетики и информационных технологий
Кафедра «Программной инженерии»

Юсупов Рашид

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

На соискание академической степени магистра

Название диссертации	Создание библиотеки компонентов визуализации аналитических данных на базе Framework Angular и технологии SVG
Направление подготовки	6М100200 – Системы информационной безопасности

Научный руководитель
доктор тех. наук
_____А. Мухамедгалиев
«27» Июнь 2020 г.

Оппонент
канд. тех. наук, профессор
_____ Н.Т. Дузбаев
«02» Июль 2020 г.

ДОПУЩЕН К ЗАЩИТЕ
Заведующий кафедрой ПИ
Доктор PhD
_____ М. Тұрдалыұлы
«__»_____ 2020 г.

Нормоконтроль
лектор
_____ Ж.М.Алибиева
«__» _____ 2020 г.

Алматы 2020

Институт кибернетики и информационных технологий

Кафедра Программная инженерия

Специальность: 6М100200 – Системы информационной безопасности

УТВЕРЖДАЮ
Заведующий кафедрой ПИ
Доктор PhD
_____ М. Тұрдалыұлы
«___» _____ 2020 г.

ЗАДАНИЕ
на выполнение магистерской диссертации

магистранту Юсупову Рашиду

Тема диссертации: «Создание библиотеки компонентов визуализации аналитических данных на базе Framework Angular и технологии SVG»
Срок сдачи законченной диссертации «___» _____

Исходные данные к магистерской диссертации. Дана информация о некоторых аналитичеких данных. Постановленные цели и задачи диссертационной работы направлены на создание библиотеки компонентов для визуализации полученных данных.

Перечень подлежащих разработке в магистерской диссертации вопросов или краткое содержание магистерской диссертации: а) определение требований – анализ целей, задач и назначения разработки; б) исследование подхода, реализуемого в библиотеке компонентов для визуализации аналитических данных; в) предобработка полученных аналитических данных; г) создание компонента в библиотеке; д) эксперименты, по оценке качества библиотеки компонентов визуализации данных.

ГРАФИК
подготовки магистерской диссертации

Наименование разделов, перечень разрабатываемых вопросов	Сроки представления научному руководителю	Примечание
Раздел 1. Общее описание		
Раздел 2. Функциональность		
Раздел 3. Экспериментальная часть		

Консультации по проекту с указанием относящихся к ним разделов проекта

Раздел	Консультант, (уч. степень, звание)	Сроки	Подпись
Нормоконтроль	Ж.М.Алибиева, Лектор кафедры программная инженерия		

Дата выдачи задания "____"_____ 2020 г.

Заведующий кафедрой _____ М. Тұрдалыұлы

Научный руководитель _____ А. Мухамедгалиев

Задание принял к исполнению магистрант _____ Р. Юсупов

Дата "_____"_____ 2020 г.

АННОТАЦИЯ

Данная работа предназначена для формирования различных графиков и визуального отображения аналитических данных.

Предлагаемое решение позволяет разработчикам использовать готовые библиотеки для визуализации аналитических данных.

В этой работе была собрана информация о способе создания библиотеки и использования технологии SVG.

Целью данной работы является визуализация аналитических данных и создание библиотеки для сокращения копирования кода.

Разработка данного проекта была реализована с применением Framework Angular и технологии SVG.

АҢДАТПА

Бұл жұмыс әртүрлі графиктерді құруға және аналитикалық мәліметтерді визуалды түрде көрсетуге арналған.

Ұсынылған шешім әзірлеушілерге аналитикалық мәліметтерді визуализациялау үшін дайын кітапханаларды пайдалануға мүмкіндік береді.

Бұл жұмыста кітапхана құру және SVG технологиясын пайдалану туралы ақпарат жиналды.

Бұл жұмыстың мақсаты - аналитикалық мәліметтерді визуализациялау және код көшірмесін азайту үшін кітапхана құру.

Бұл жобаның дамуы Angular Framework және SVG технологиясы бойынша жүзеге асырылды.

SUMMARY

This work is intended for the formation of various graphs and visual display of analytical data.

The proposed solution allows developers to use ready-made libraries for visualization of analytical data.

In this work, information was collected on how to create a library and use SVG technology.

The aim of this work is to visualize analytical data and create a library to reduce code copying.

The development of this project was implemented using Angular Framework and SVG technology.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На данный момент, во время быстрого развития информационных технологий разработчикам зачастую приходится визуализировать различные данные на графиках. Когда проектов становится больше чем один, возникает необходимость повторно использовать не только отдельные модули с кодом, но также приходится копировать компоненты, предназначенные для визуализации аналитических данных.

Существует большое количество решений данной проблемы, начиная от традиционного копирования кода или компонентов, до настройки отдельного проекта. Проблема в том, что это занимает значительные усилия по подготовке и каждый такой проект уникальный со своим инструментарием и каждому новому разработчику придется разбираться заново.

Данный проект облегчит визуализацию аналитических данных и даст возможность прорисовать эти данные на различных графиках, а также позволит визуализировать аналитические данные не копируя код.

1 Общее описание

1.1 Цель разработки приложения

Созданные библиотеки для визуализации аналитических данных представляют собой компоненты, которые позволяют разработчикам удобно отображать различные аналитические данные в виде разных графиков и карт.

Преимущества данных библиотек для отображения аналитических данных заключается в более высокой точности и удобности выполнения работы разработчика, по сравнению с тем что разработчик будет создавать график или карту самостоятельно, а также более высокой скоростью, так как нет необходимости создавать самостоятельно различные графики и карты вручную разработчикам компании, так же в коде разработчика пропадает копирование одного и того же кода, либо исчезает надобность создания нового проекта для визуализации аналитических данных, что существенно облегчает выполнение работы и значительно ускоряет его.

Разработчик компании установивший данную библиотеку может отображать различные аналитические данные предварительно обработав их после получения из базы, также имеет возможность динамично менять отображение аналитических данных во время их изменения, что увеличивает удобство работы сотрудника, и позволяет более удобно увидеть данные в графическом виде пользователю.

Данные библиотеки так же имеют возможность взаимодействия с пользователем при нажатии различных элементов графика либо карты. Разработчик получает события при нажатии на различные элементы графика либо карты и в дальнейшем может их обработать исходя из логики проекта, и это позволяет еще более удобно управлять данными отображая их на графике.

Использующие библиотеки предусматривают отсутствие данных при их использовании, что в свою очередь, минимизирует количество ошибок и нарушения отображения аналитических данных на графике или карте, так же не нарушается прорисовка графика.

В связи с использованием технологии SVG, данные графики имеют возможность удобного масштабирования без потери качества, что позволяет отображать графики или карты на различных размерах экрана и при любых изменениях окна браузера, что в свою очередь, увеличивается удобность использования, так как разработчику не придется создавать различные размеры контейнера для использованных графиков либо карт из библиотеки, и это ускоряет выполнения работы.

1.2 Определения, термины и сокращения

В таблице 1 сформулированы все сокращения и термины, которые были использованы в предметной области разрабатываемого проекта, так же термины, связанные с программной реализацией проекта и используемыми технологиями при разработке.

Таблица 1 – Термины, Сокращения и их определения

Термины и сокращения	Определение
SVG	Scalable Vector Graphics
CSS	Cascading Style Sheets
SASS	Syntactically Awesome Stylesheets
CLI	Command Line Interface
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
HTML	HyperText Markup Language
DOM	Document Object Model
SPA	Single Page Application
XML	eXtensible Markup Language

1.3 Обоснование выбора программного обеспечения

Во Время разработки автор данного проекта столкнулся со следующими проблемами:

1) Создание библиотеки;
2) Масштабирование графика;
3) Динамичность графика;
4) Возвращение события в ответ на нажатие на графики;
5) Создание нескольких графиков в одной библиотеке;
6) Обработка данных для прорисовки графика;

Чтобы решить все эти проблемы, автор данного проекта использовал следующее решение:

1) Angular;
2) SVG;

1.3.1 Angular

Автор данного проекта использовал Angular в связи с тем, что в нем достаточно особенностей для решения со столкнувшимися проблемами.

Angular является Фреймворком, который разработала компания Google, в области применения для разработки приложений клиентской части.

Для начала он нацелен на создание SPA решений (Single-Page-Application), другими словами одностраничных приложений.

Angular дает возможность такой функциональности, как связывание двух сторон, которое позволяет динамически менять данные в одном элементе интерфейса при замене данных модели в другом, маршрутизация, шаблоны и другие [1].

Angular library

Angular library позволяет создавать и публиковать новые библиотеки для расширения функциональности Angular. Это дает возможность решить одну и туже задачу в нескольких приложениях. Библиотека включает в себя повторно используемый код, который определяет компоненты, сервисы и т.д. Библиотека упакована в пакет npm для публикации и совместного использования, и этот пакет может также включать схемы, которые предоставляют инструкции для генерации ng generate component [2].

Компоненты

Компонент позволяет управлять отображением на экране представления, по умолчанию для ее основы используется Shadow DOM (для того что бы создать инкапсулированного визуального поведения).

В основном компоненты могут использоваться во время создания обычного виджета в пользовательском экране, так же они могут показывать некоторый набор еще более обычных компонентов внутри себя (для того что бы увеличить абстракции и разработку обычных функциональных виджетов внутри какого-либо приложения).

Шаблоны

Шаблон представляет собой HTML разметку, в которой есть возможность описания вашего взаимодействия с DOM основываясь на модели данных и событий класса компонента (для примера, контроллер MyComponent).

Внедрение зависимостей

Внедрение зависимостей представляет собой композицию структурных шаблонов для проектирования, при которой за любую функцию самого приложения отвечает один, независимый объект (сервис), который имеет необходимость для использования других объектов (зависимости), интерфейсы которые ему известны.

Зависимости могут передаваться (внедряются) сервису во время его создания.

Дерективы

Директива предоставляет возможность получения прямого доступа к DOM элементов.

Они могут быть двух видов: атрибутные и структурные.

Атрибутная директива:

@Directive({

```
        selector: '[bold]'
    })
    export class BoldDirective {
        constructor(private elementRef: ElementRef){
            this.elementRef.nativeElement.style.fontWeight = "bold";
        }
    }
```

Тут будет внедрен сервис "ElementRef".

Он показывет ссылку на часть, к которой в дальнейшем будет применена директива:

```
<p bold>Hello world</p>
```

Структурные директивы:

Структурные директивы имею возможность изменения структуры DOM при помощи удаления и добавления html элементов.

На данный момент есть три структурной директивы: ngFor, ngIf и ngSwitch.

1.3.2 SVG

SVG - Scalable Vector Graphics это разметка, основанная на XML, который содержит двумерные векторы. Векторами могут быть простые геометрические формы, сложные контуры, да и всё то же самое, что можно сделать в Иллюстраторе. Этот формат изображений имеет намного больше общего с веб-страницей, чем тот же JPEG. SVG намного мощнее — им легко можно управлять при помощи кода.

Подготовка и оптимизация SVG.

Подготовка SVG для использования в вебе это очень простой процесс, не сложнее экспорта JPEG или PNG. Можно использовать любой графический редактор (Illustrator, Corel Draw, Sketch, Inkscape, с тем размером изображения, который вы планируете использовать. Следует перевести текст в кривые, поскольку шрифт, скорее всего, будет неправильно отображаться, либо их можно стилизовать с помощью веб-шрифта, используемого на странице. Не стоит также превращать все объекты в единые формы, особенно если у вас есть обводка, которой необходимо будет управлять на странице, тем более преобразование объектов зачастую не уменьшает размер файла. Любые имена, присвоенные группам или слоям, будут добавлены к SVG как ID элемента. Это довольно удобно для стилизации, но немного увеличит общий размер файла.

Перед тем как сделать экспорт, необходимо проверить, все ли изображения находятся в целочисленной пиксельной сетке. В противном случае скорее всего изображению не будет хватать чёткости и часть изображения обрежется.

Варианты реализации.

IMG

Здесь всё делается так же, как с любым изображением в этом формате. Можно даже использовать SVG как элемент <picture>.

<img src="image.svg" alt="Title for image" height="50" width="55">

Background-image

Не стоит сохранять этот формат в base64, это приведёт к блокировке загрузки стилей.

```
.logo {
        background-image: url(image.svg);
}
```

Iframe

Вы можете загрузить SVG как <iframe>.

<iframe src="image.svg">Iframe</iframe>

Object

<object> Лучший вариант, позволяет изменять SVG, не встраивая его в HTML.

<object type="image/svg+xml" data="image.svg">SVG </object>

SVG

Данный вариант позволяет самому написать нужный дизайн того или иного графика и в любой момент изменить его как нужно программисту. Он имеет различные теги для рисования разных фигур.

<svg> … </svg>

CSS – Манипуляция SVG

SVG, предоставляет возможность изменять стили его элементов, используя CSS. К примеру, нам потребуется определить столбцы разными цветами для отображения аналитических данных, на одной и той же странице это можно сделать, не создавая нового столбца.

Есть два способа изменить стили — во встроенном SVG и через внешнее подключение. Чтобы встроить стили, нужно обернуть их в тег <style> и также внутри <! [CDATA [...]]>. Иногда XML анализаторы могут конфликтовать с определённым символами (например>). Желательно использовать CDATA, так как данный конфликт может сломать все. В основном встроенные стили будут работать со всеми реализациями <img> и background-image.

При использовании внешних стилей, нельзя использовать <img> или background-image. При использовании <object>, нужно добавить сноски к списку стилей, внутри SVG-файла.

Фиксированная ширина и адаптивность

При использовании изображений в адаптивном дизайне существуют две опции, которые помогают стилизовать их: можно использовать фиксированные размеры и регулировать контрольные точки по необходимости или позволить им изменять размер на странице в зависимости от родительского контейнера.

При фиксированных размерах и использовании SVG как background-image следует прописывать background-size, так как браузеры могут подрезать изображение, или сжать его, особенно, если отображаются размеры, отличные от оригинального размера изображения [3].

2 Функциональность

2.1 Функции системы

Библиотека дает возможность разработчику более удобно и быстро визуализировать аналитические данные в виде графиков и карт, не разрабатывая собственный код, и при этом экономя время.

Также библиотека может динамично отображать аналитические данные, принимая различные события и перерисовывая график либо меняя данные в карте.

Библиотека имеет возможность передавать в использующий его компонент различные события в ответ на клики на самом графике либо карте.

Все полученные аналитические данные предварительно обрабатываются для визуализации их в виде графиков либо отображения их на карте.

Технология, которая позволит создать библиотеку будет Angular. Данная технология позволяет создать библиотеку которая будет визуализировать аналитические данные более 1 графика.

Для создания графиков либо карт для визуализации будет использована технология SVG. Данная технология позволяет прорисовывать дизайн для графиков. Также эта технология позволяет масштабировать графики либо карты в любой размер. При изменении размеров окна браузера либо при отображении графиков на различных мониторах, от маленьких до больших, не теряется качество изображения прорисованного графика либо карты, так же при масштабировании не меняется сам созданный дизайн разработанного графика либо карты. Также вспомогательным элементом был использован CSS, для разработки различного дизайна для графиков либо карт.

Для разработки функциональности, обработки полученных аналитических данных, отправка события в компонент при нажатии пользователем на график, расчеты для визуализации был использован язык программирования Typescript.

Для предоставления доступа разработчикам к данным библиотекам, для визуализации аналитических данных был использован менеджер пакетов npmjs, входящий в состав Node.js.

2.2 Описание графиков

В рамках Института цифровой техники и технологии, для данной библиотеки были созданы следующие графики визуализации аналитических данных:

1) Круговой график по расчету реализаций жилья БД;
2) Круговой график для распределения текущих средств;
3) Круговой график с картой для визуализации данных;
4) Карта для отображения аналитических данных;

Круговой график по расчету реализации жилья БД

Данный график предназначен для отображения информации по реализации жилья. Реализация жилья делится на следующие виды:

1) Кредитное жилье;
2) Арендное жилье;
3) Жилье для иных лиц;

Этот график представляет из себя 2 круга:

1) Внутренний;
2) Внешний;

Внутренний круг (толстый) служит для отображения данных в графическом виде по трем видам жилья, который разделяет круг на три толстые части и показывает общее количество к реализации жилья.

Внешний круг (тонкий) служит для отображения данных в графическом виде по трем видам жилья, который делит круг на три тонкие части и показывает количество уже реализованного жилья, так же данные отображаются с процентами.

По середине графика отображаются данные в текстовом и числовом виде, которые показывают информацию об общем, реализованном и остатке жилья.

Так же данный график имеет возможность динамично меняться при любых изменениях данных, которые были получены во время его использования.

График отображает информацию по двум видам данных:

1) Квартиры, в соответствии с рисунком 2.1;
2) Деньги, в соответствии с рисунком 2.2;

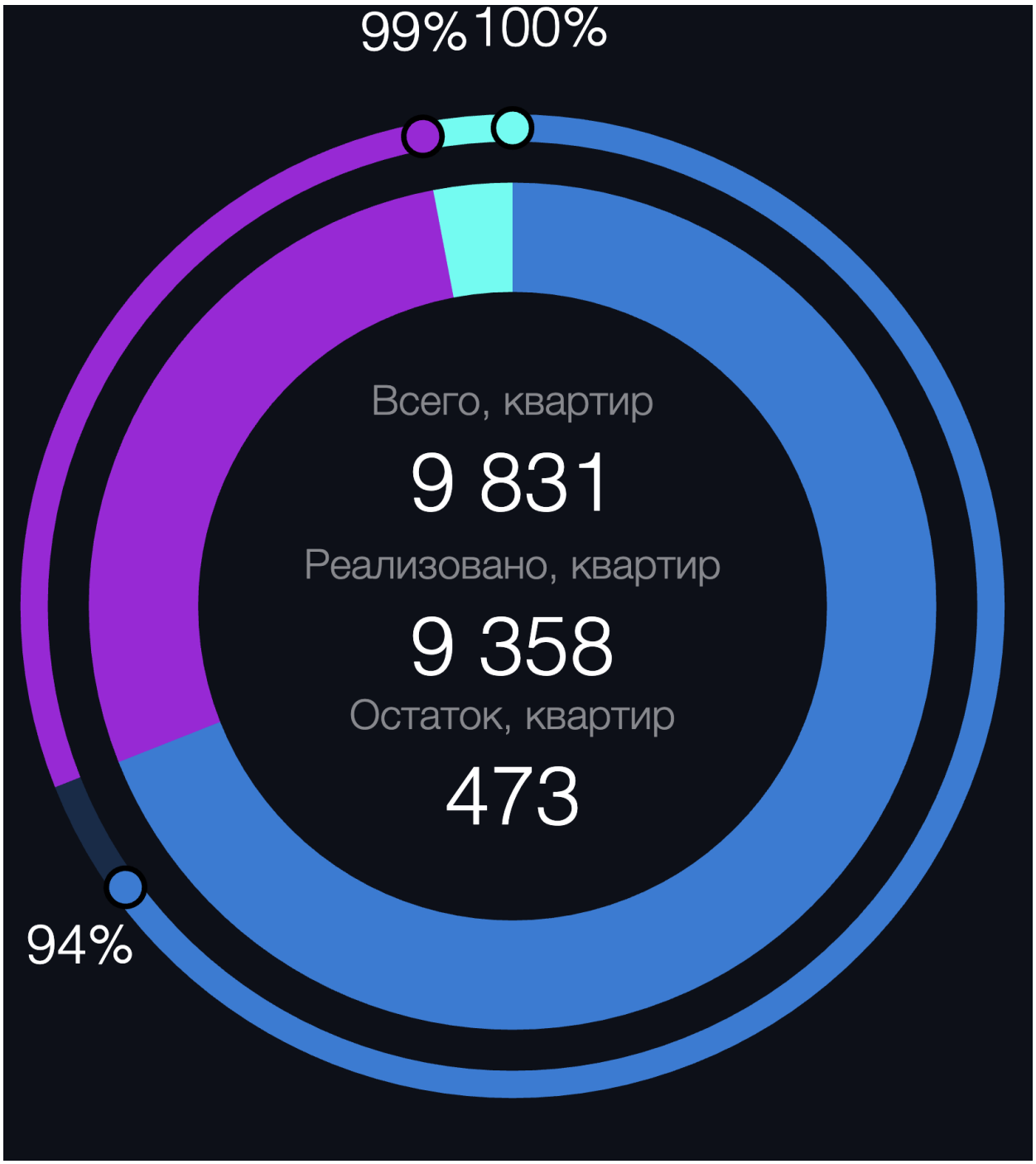


Рисунок 2.1 График реализации жилья БД по квартирам

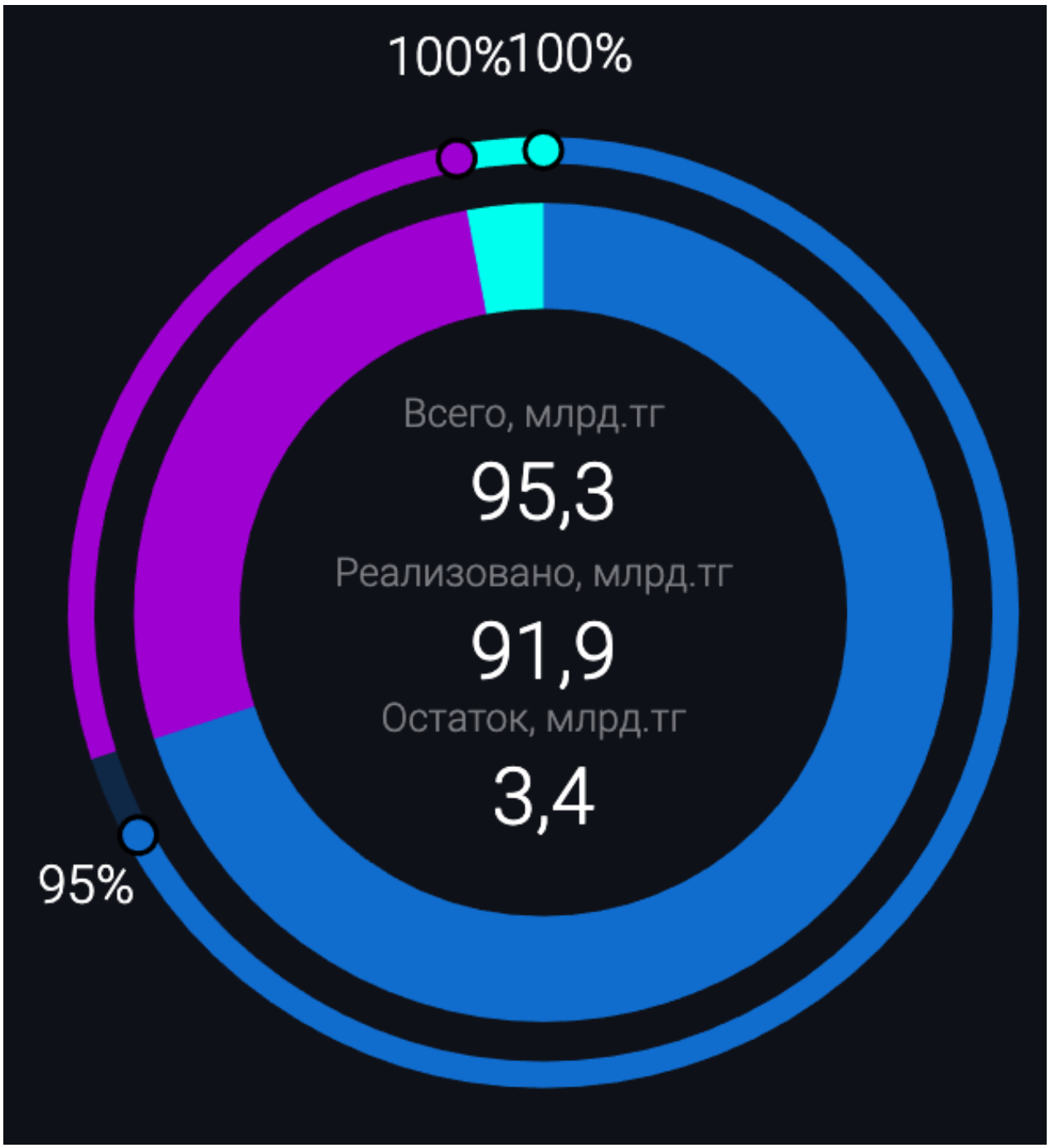


Рисунок 2.2 График реализации жилья БД в деньгах

Круговой график для распределения текущих средств

Этот график предназначен для визуализации данных по количеству потраченных денег для материала. Материалы делятся на следующие виды:

1) Сыпучие материалы:
   - Цемент;
   - Песок;
   - Прочее;
2) Бетон:
   - Тяжелый класс В25;

- Тяжелый Класс В15;
- Прочее;

3) Металл:
   - Арматура;
   - Балки;
   - Прочее;
4) Кладочные материалы:
   - Кирпич М-75;
   - Кирпич М-100;
   - Прочее;
5) Окна и двери:
   - Окна пластиковые;
   - Двери межкомнатные;
   - Прочее;
6) Отделочные материалы:
   - Теплоизоляция;
   - Крепежные материалы;
   - Прочее;
7) Сантехника, электрика;
   - Автоматы;
   - Насосы;
   - Прочее
8) Прочее:
   - Краска;
   - Клей;
   - Прочее;

Данный график состоит из восьми отдельных частей для каждого вида материала. Внутри каждой из частей отображены рисунки в соответствии с их категорией. От каждой части выведены линии для отображения информации в текстовом и числовом виде. Каждая из этих частей имеет возможность выделения, что в свою очередь возвращает события для разработчика для дальнейшей обработки. Так же данный график позволяет динамично менять отображаемые данные для пользователя. По середине графика отображается общая информация по расходу в числовом и текстовом виде.

График может быть в двух состояниях:

1) Без выделения согласно рисунку 2.3;
2) С выделением согласно рисунку 2.4;

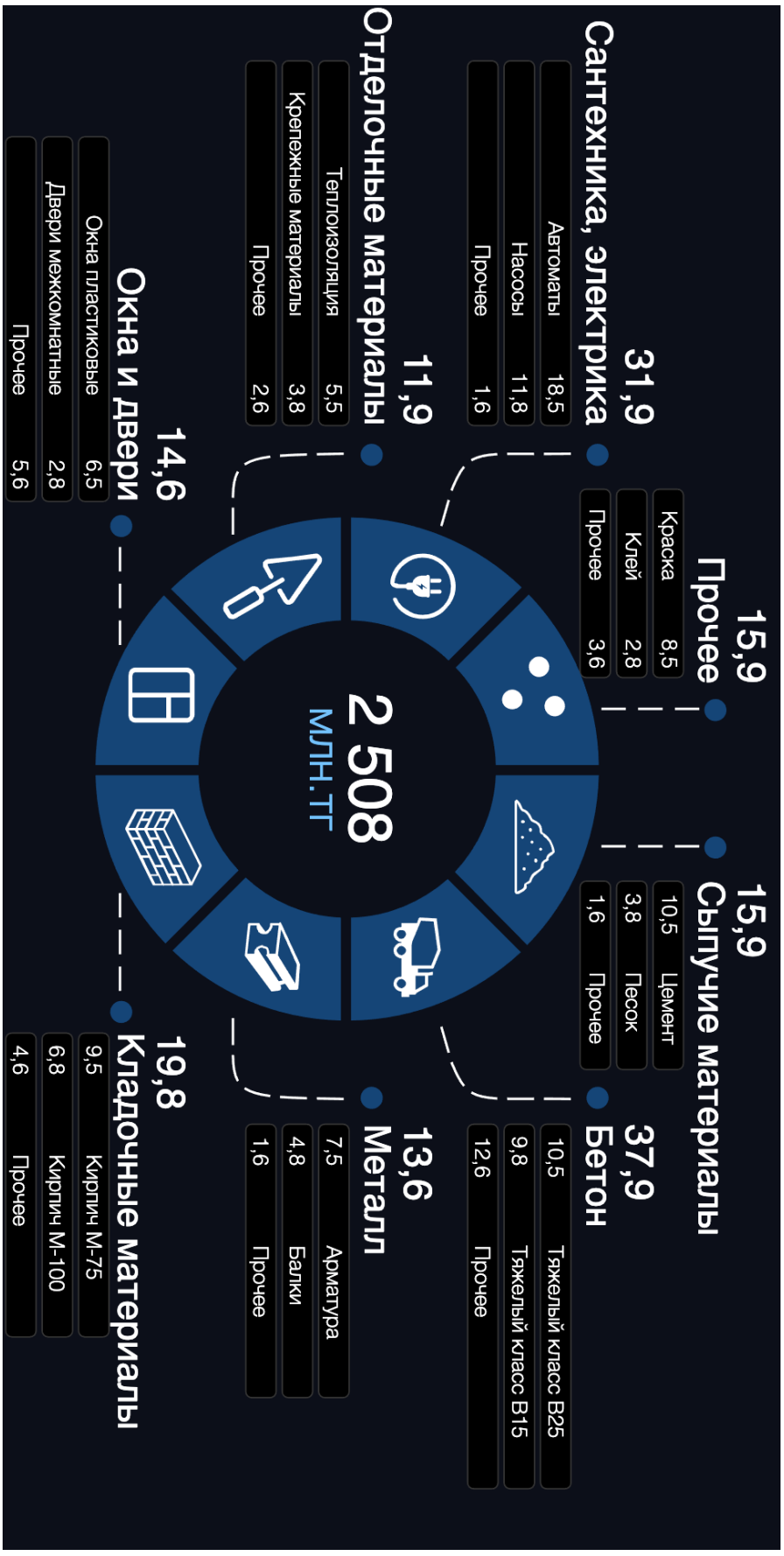


Рисунок 2.3 График для распределения текущих средств без выделения

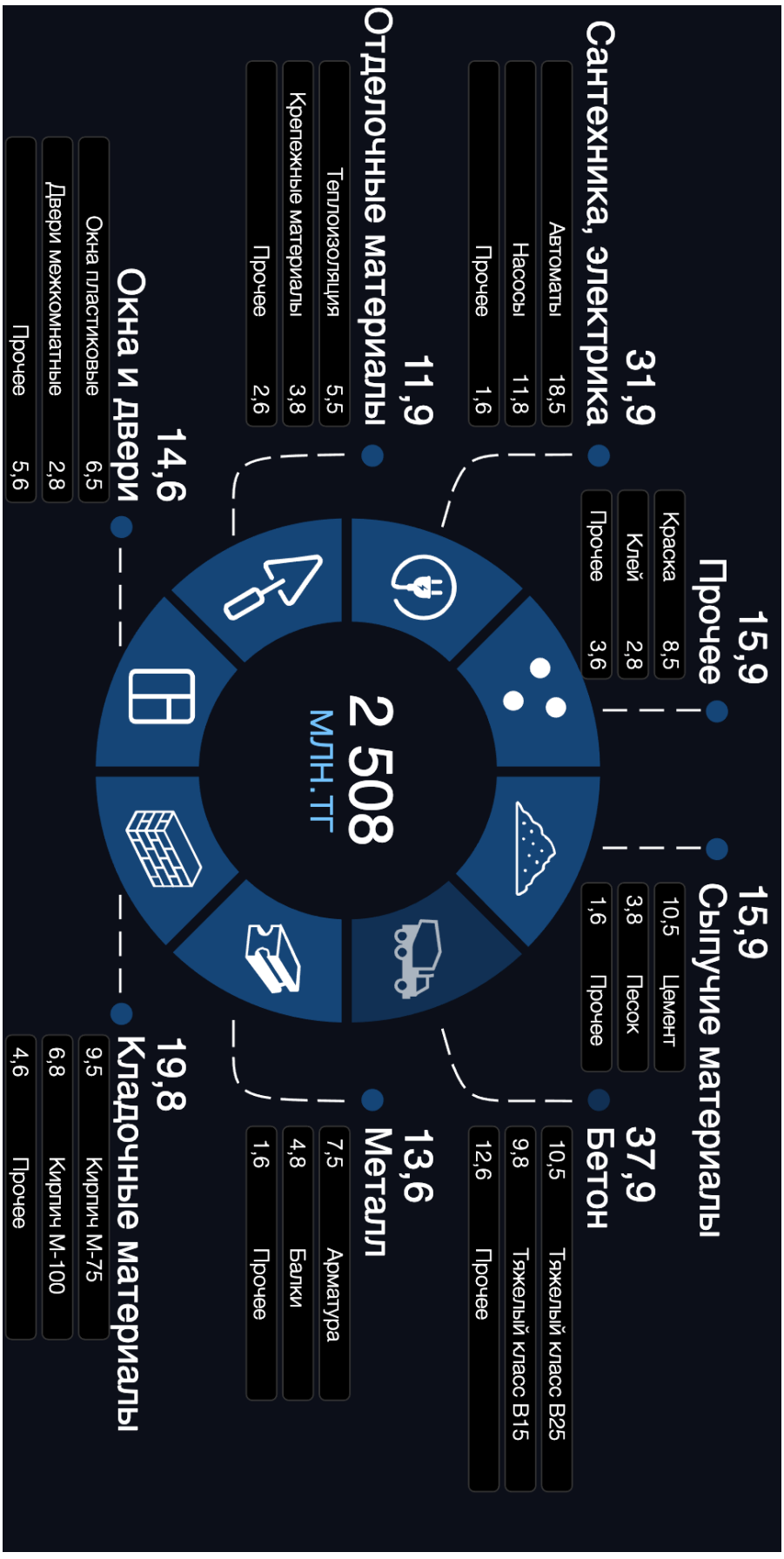


Рисунок 2.4 График для распределения текущих средств с выделением

Круговой график с картой для визуализации данных

Данный график разработан для визуализации аналитических данных по приобретению облигаций по месяцам в двух годах одновременно. Этот график представляет собой круг, который разделен на все месяца, каждого из годов в соответствии с рисунком 2.5. Верхний полукруг отображает данные за один год, нижний полукруг визуализирует информацию из полученных данных за второй год. Информация отображается исходя из полученных данных в виде плана и факта.

Каждый месяц в круговом графике выделен собственным цветом, для улучшения видимости и разборчивости. От каждого месяца выведены линии для отображения плана и факта в числовом виде. Линия плана нарисована с пунктирами, а линия факта без пунктиров.

В правой части кругового графика отведено общее количество по месяцам плана и факта, для каждого года отдельно.

В левой части данного графика отображены два выбранных года, по которым визуализируются аналитические данные. Менять выбранный год можно при нажатии на отведенные стрелки, которые отображены с лева от года. При нажатии на верхнюю стрелку года повышаются, при нажатии на нижнюю стрелку года понижаются.

В центре всего кругового графика отображена карта Республики Казахстан. Каждая область на карте подписана своим названием в текстовом виде. Кроме областей на карте отображены и города Республики Казахстан.

От каждого месяца, до областей на карте прорисованы линии, пунктирные для плана и без пунктиров для факта. Линия от месяца до области либо города прорисовывается только в том случае, если в определенной области есть данные за какой-либо месяц. Для более красивого отображения, линии от месяца к городам или областям рисуются с изгибами.

В левом нижнем углу отображены два чек-бокса, которые подписаны в текстовом виде, как план и факт и под каждой надписью прорисованы линии, с пунктирами для плана и без пунктиров для факта, для того что бы пользователь понимал какая из линий к чему относится, они служат для выбора отображения плана либо факта прорисованных линий от месяца к областям, либо городам.

Каждая область либо город могут быть выделены. При выделении области либо города, выбранная часть меняет цвет для того что бы пользователь понимал, что область была выделена. Так же после выделения пропадают все линии, прорисованные от всех месяцев ко всем областям, кроме тех линий, которые ведут к выбранной области либо города.

Каждый месяц из обоих годов так же может быть выбран. При выборе любого из месяцев, выделенная часть остается подсвеченной, остальные месяцы затемняются, для понимания для пользователя, что месяц был выбран. Так же после выделения любого из месяцев линии прорисованные до областей или городов становятся не видимыми, кроме тех линий, которые проведены от выделенного месяца.

Одновременно можно выделить лишь один месяц из всего круга, либо один город или область.

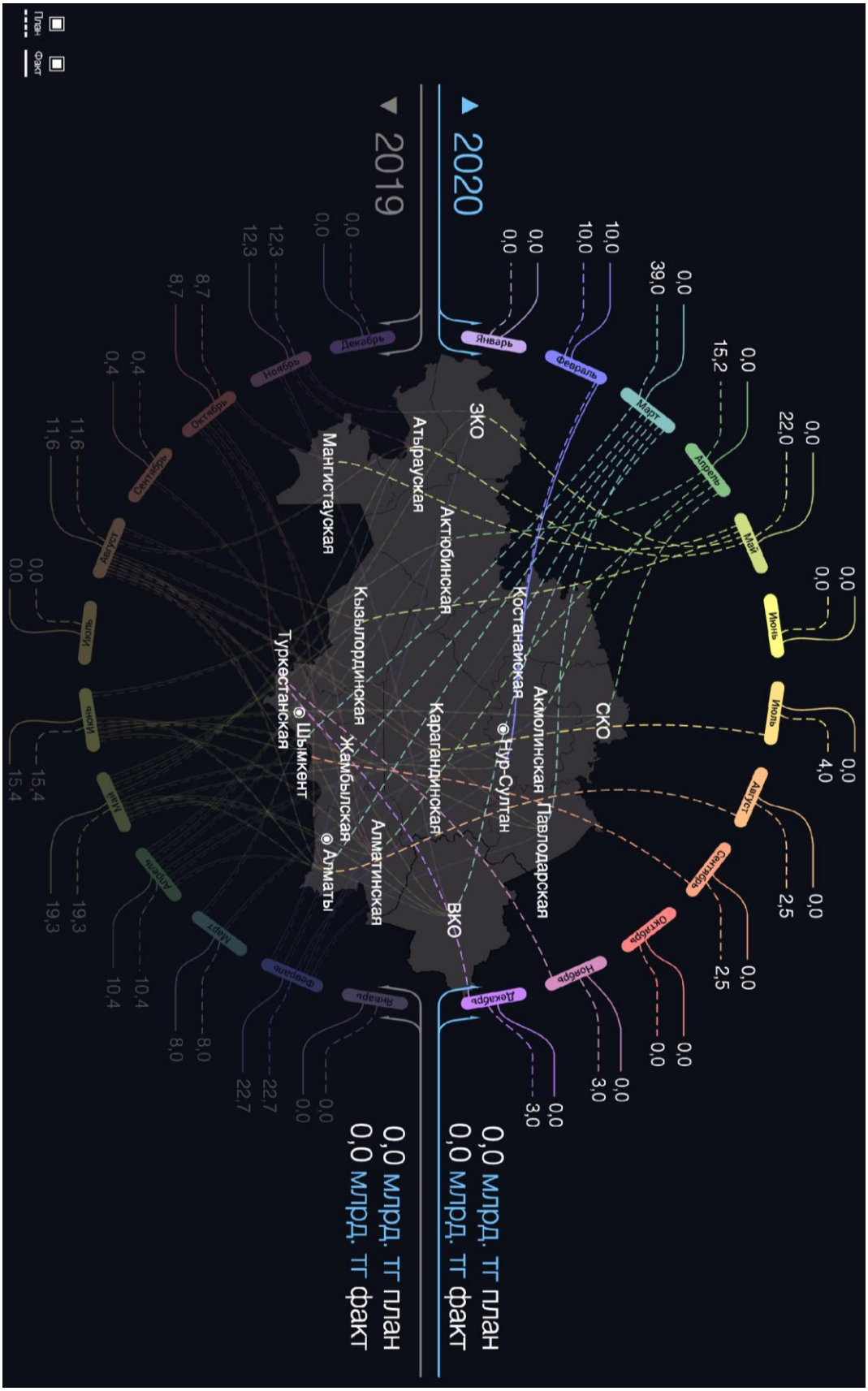


Рисунок 2.5 График приобретения облигаций по месяцам

Данный круговой график приобретения облигация по месяцам может быть выделен один из месяцев любого из годов как показано на рисунке 2.6.

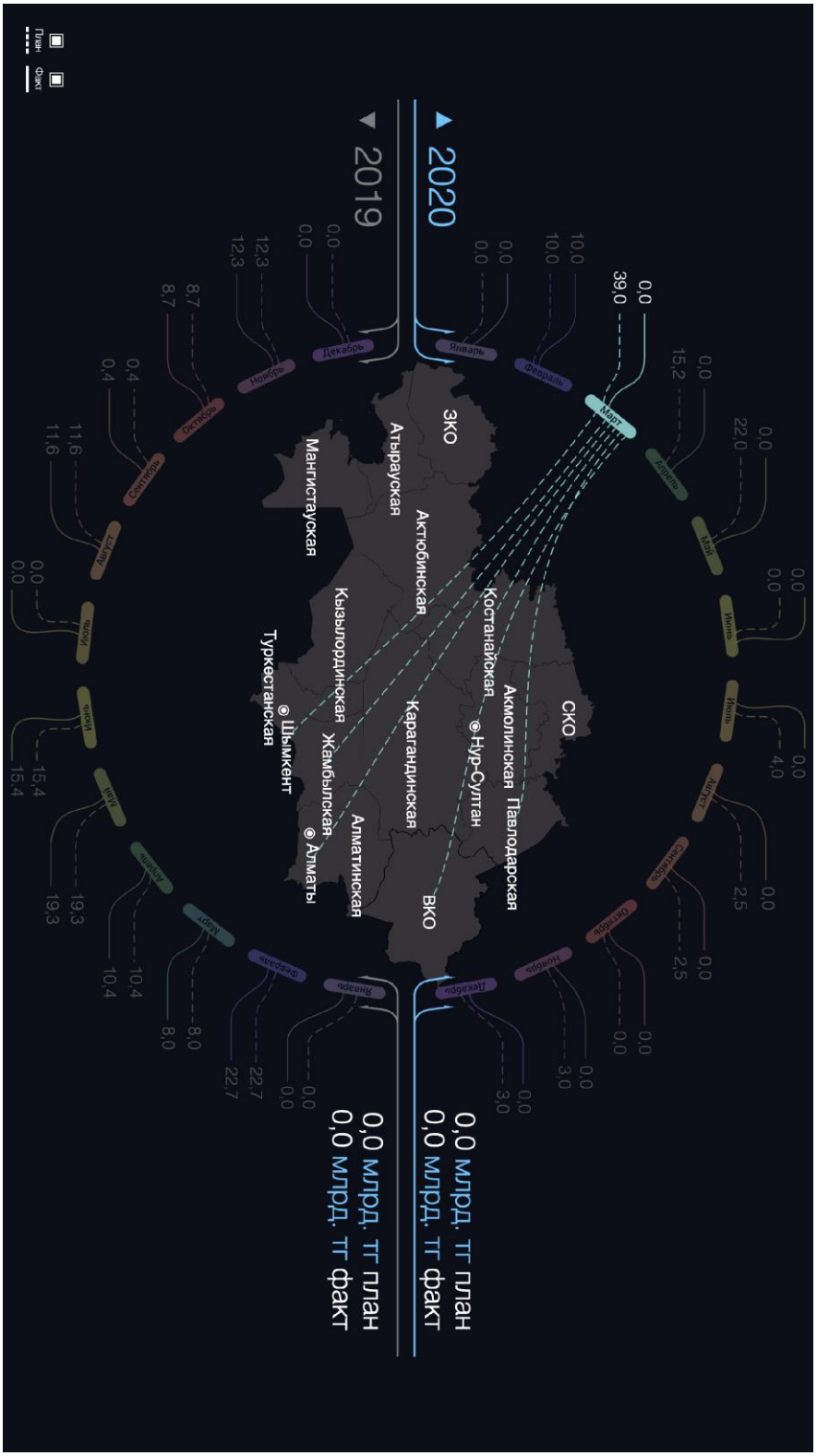


Рисунок 2.6 График приобретения облигаций с выделенным месяцем

Так же этот график имеет возможность выделения области либо города из карты при котором остаются видными только те линии, которые прорисованы от различных месяцев до выделенной области в соответствии с рисунком 2.7.

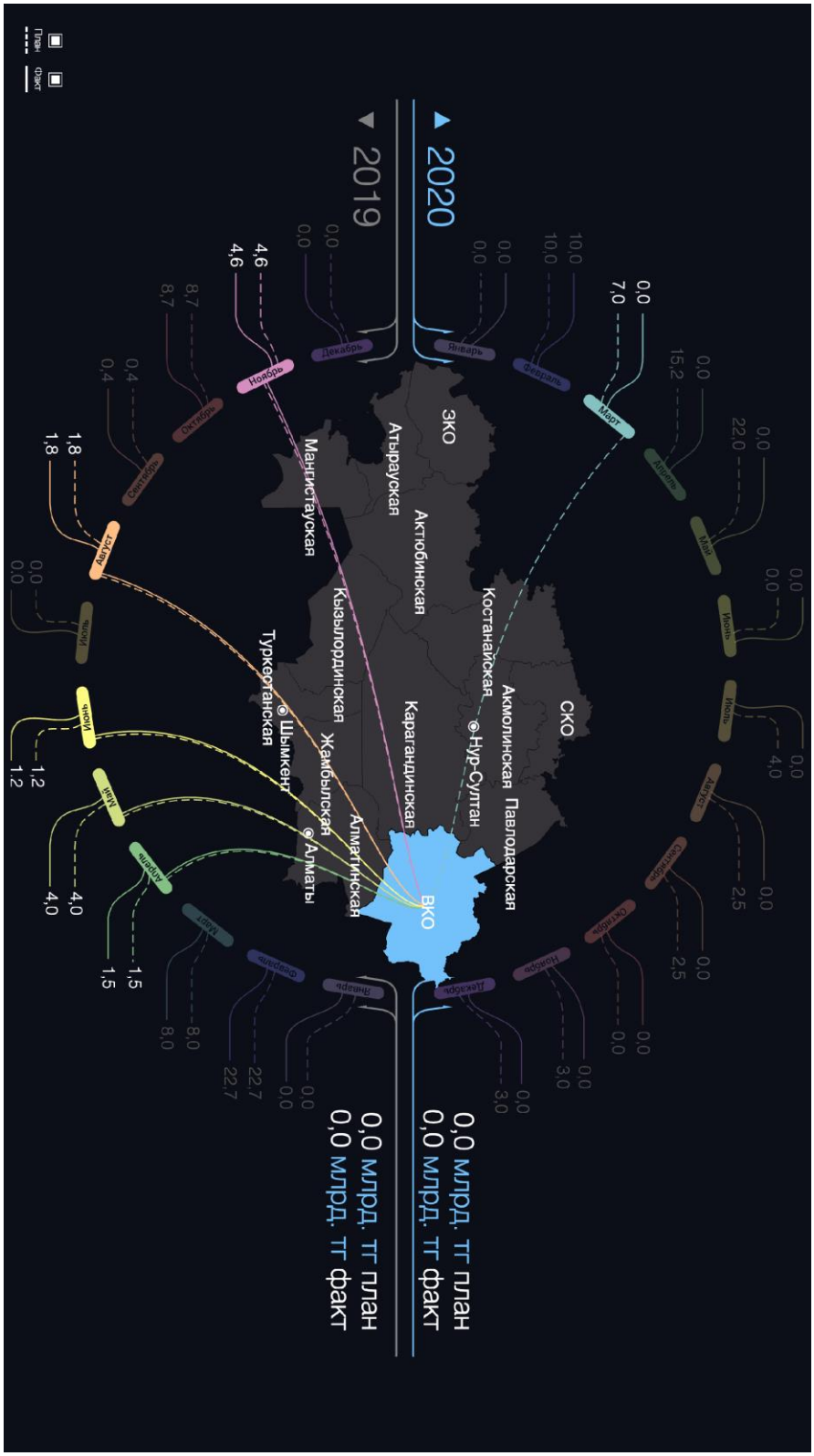


Рисунок 2.7 График приобретения облигаций с выделенной областью

Карта для отображения аналитических данных

Этот график предназначен для отображения различной числовой информации по областям либо городам.

Он представляет из себя карту Республики Казахстан, на которой выделена каждая область. Каждая область подписана в текстовом виде. Так же на данной карте кроме областей есть города, тоже подписанные в текстовом виде.

Отображение числовой информации на карте предусмотрено немного выше надписи каждой области либо города.

Данная карта имеет возможность выделения любой из областей или городов, отображенных на ней. При выделении одной из области либо города, выделенная часть становится темнее остальных. Так же после выделения любой из области или города, в компонент использующий эту карту отправляется событие с id выбранной области. Далее разработчик может обработать полученные данные.

Эта карта так же может менять цвета своих областей. Для того что бы изменить цвет карты, разработчику в массиве с данными нужно отправить код цвета. В Зависимости от кода цвета, будет меняться и цвет карты. Так же карта может динамично менять как цвета, так и информацию, отображенную на карте.

Карта имеет два состояния:

1) С выделенной областью либо городом в соответствии с рисунком 2.8;
2) Без выделенной области либо городом в соответствии с рисунком 2.9;

Рисунок 2.8 Карта Республики Казахстан с выделенной областью

Рисунок 2.9 Карта Республики Казахстан

3 Экспериментальная часть

3.1 Создание библиотеки для визуализации аналитических данных

Первый этап – это создание каркаса, которая будет служить для разработки в нем различных библиотек, для этого напишем команду в командной строке, в созданной вами директории:

ng new BaiterekLibrary

Данная команда создаст проект Angular Library в созданной вами папке.

Вторым этапом после создания каркаса нам нужно создать саму библиотеку, для этого перейдем в корневой каталог нашего каркаса и введем команду:

ng generate library baiterek-lib

Данная команда создаст в проекте *BaiterekLibrary* библиотеку baiterek-lib, у которой будет свой модуль.

И так у нас создан каркас для библиотек, создан модуль с библиотекой в котором мы можем создавать компоненты для визуализации аналитических данных.

3.2 Круговой график по расчету реализации жилья БД

Первым этапом разработки компонента для визуализации аналитических данных, будет создание самого компонента в каталоге Libraries в проекте BaiterekLibraries, для этого пишем команду:

ng generate component pie-chart

Данный компонент будет служить для визуализации аналитических данных по квартирам, сколько всего квартир, сколько реализовано и остаток.

Второй этап – это разработка дизайна данного графика. Первым делом нам нужно создать внутренний толстый круг, и внешний тонкий круг нашего графика. У этих кругов должны отображаться 3 вида данных:

1) Кредитное жилье;
2) Арендное жилье;
3) Жилье для иных лиц;

Данная информация по видам данных должна отображаться в этих кругах в сто процентной шкале и каждый вид данных будет занимать свой процент и цвет. Для этого создадим массив и опишем его в классе PieItem как показано на рисунке 3.1

Рисунок 3.1 Класс PieItem

Данный класс описывает что каждый объект в массиве имеет свой цвет (color), который служит для отображения определенного цвета для каждого из видов полученных данных, так же процент (percent), для получения процента, внутренний толстый круг (dasharray1) и внешний тонкий круг (dasharray2).

Третий этап – это создание html страницы. Для визуализации внутреннего и внешнего кругов нам нужно написать следующий фрагмент кода в соответствии с рисунком 3.2

Рисунок 3.2 html код для визуализации внутреннего и внешнего кругов

Четвертым этапом будет реализация заполнение процентов внешнего круга. То есть это будет показатель на сколько было реализовано по квартирам того или иного вида жилья. Для этого создадим еще один массив и опишем его в классе PieOuterItems в соответствии с рисунком 3.3.

```
You, 2 months ago | 1 author (You)
export interface PieOuterItem {
  color: string;
  percent: number;
  percentText: number;
  opacity: number;
  circle: PieCircle;
  dasharray?: string;
  rotate?: number;
  textCoordinates?: { x: number, y: number };
}
```

Рисунок 3.3 Класс PieOuterItems

Пятым этап – это создание html кода для отображения заполнения процентов внешнего круга. Для этого следует написать следующий фрагмент кода как показано на рисунке 3.4.

```
<g *ngFor="let item of newOuterItems; let i = index">
  <g [attr.transform]="offset(i)">
    <circle *ngIf="item.percent == 0 ? false : true" cx="175" cy="0" r="7" [attr.stroke]="item.circle.stroke"
      [attr.fill]="item.circle.fill" stroke-width="2" [attr.transform]="'rotate(' + item.rotate + ')'"></circle>
  </g>
  <text *ngIf="item.percent == 0 ? false : true" class="percent" [attr.fill]="'#ffffff'" text-anchor="middle"
    [attr.x]="item.textCoordinates.x" [attr.y]="item.textCoordinates.y" [attr.transform]="'rotate(' + 90 + ')'">
    {{ item.percentText }}%
  </text>
</g>
```

Рисунок 3.4 html код для заполнения процентов внешнего круга

Шестым этап – это разработка текста и числовых данных в центре графика для отображения следующих данных:

1) Всего;
2) Реализовано;
3) Остаток;

Для этого напишем следующий фрагмент кода в соответствии с рисунком 3.5.

```
<g *ngIf="textValue !== null">
  <text class="value" transform="rotate(90)" y="-30" text-anchor="middle">
    <tspan class="lbl" y="-70" x="0">Всего, {{textValue}}</tspan>
    <tspan y="-35" x="0">{{totalFlatCoint}}</tspan>
  </text>

  <text class="value" transform="rotate(90)" y="35" text-anchor="middle">
    <tspan class="lbl" y="-10" x="0">Реализовано, {{textValue}}</tspan>
    <tspan y="25" x="0">{{saleFlatCount}}</tspan>
  </text>

  <text class="value" transform="rotate(90)" y="90" text-anchor="middle">
    <tspan class="lbl" y="45" x="0">Остаток, {{textValue}}</tspan>
    <tspan y="80" x="0">{{residueFlatCount}}</tspan>
  </text>
</g>
```

Рисунок 3.5 html код для отображения текста в центе графика

Последним этапом будет создание функция для работоспособности данного графика.

Функция для обновления данных на графике при динамичных изменениях данных. Для разработки данной функции напишем следующий фрагмент кода в соответствии с рисунком 3.6.

```
ngOnChanges(simple: SimpleChanges) {
  this.updateData();
}

updateData() {
  this.newOuterItems = [...this.outerItems];
  for (let i = 0; i < this.items.length; i++) {
    let p = 0;
    if (this.items[i].percent !== 0) {
      p = (this.outerItems[i].percent / this.items[i].percent) * 100;
    }
    this.newOuterItems[i] = { ...this.newOuterItems[i], percent: p };
    this.items[i].dasharray1 = this.dashArray(175, this.items[i].percent);
    this.items[i].dasharray2 = this.dashArray(135, this.items[i].percent);
    this.newOuterItems[i].dasharray = this.dashArrayForOuterItems(175, p, i);
    this.newOuterItems[i].rotate = this.getAngleForOuterItems(p, i);
    this.newOuterItems[i].textCoordinates = this.gCoordinateForTextPercent(this.outerItems[i].percent, i);
  }
  // console.log('Items: ', this.items);
  // console.log('Outer items: ', this.outerItems);
  // console.log('New outer items: ', this.newOuterItems);
}
```

Рисунок 3.6 Функция для обновления данных

Функции для округления процентных линий и получения конца каждого из процентов. Фрагмент кода для данной функции отображен на рисунке 3.7.

```
dashArray(radius, percent) {
  return this.service.dashArray(radius, percent);
}

dashArrayForOuterItems(radius, percent, i) {
  return this.service.dashArrayForOuterItems(radius, percent, this.outerItems[i].percent);
}

offset(index) {      You, 2 months ago • pieChart complete
  return this.service.offset(index, this.items);
}

getAngle(percent) {
  return this.service.getAngle(percent);
}

getAngleForOuterItems(percent, i) {
  return this.service.getAngleForOuterItems(percent, this.items[i].percent);
}
```

Рисунок 3.7 Функции для округления графика и нахождения конца линий

Функция для получения координат для текстового отображения процентов графика. Данный фрагмент кода отображен в соответствии с рисунком 3.8

```
gCoordinateForTextPercent(percent, i) {
  if (i === 0) {
    const angle = 90 + this.getAngle(percent - 1);
    this.prevAngle = 90 + this.getAngle(this.items[0].percent);
    const x = -205 * Math.cos(angle / 180 * Math.PI);
    const y = -205 * Math.sin(angle / 180 * Math.PI);
    this.percentX = x;
    this.percentY = y;
    this.percentCreditX = x;
    this.percentCreditY = y;
    const p = { x, y };
    return p;
  }
  if (i === 1) {
    let angle = this.getAngle(percent + 1);
    angle = angle + this.prevAngle;
    const x = (-205 * Math.cos(angle / 180 * Math.PI)) - 10;
    const y = -205 * Math.sin(angle / 180 * Math.PI);
    const p = { x, y };
    this.percentRentalX = x;
    this.percentRentalY = y;
    this.prev1Angle = 90 + (this.getAngle(this.items[1].percent) + this.getAngle(this.items[0].percent));
    return p;
  }
  if (i === 2) {
    let angle = this.getAngle(percent);
    angle = angle + this.prev1Angle;
    const x = (-205 * Math.cos(angle / 180 * Math.PI)) + 10;
    const y = -205 * Math.sin(angle / 180 * Math.PI);
    const p = { x, y };
    return p;
  }
}
```

Рисунок 3.8 Функция нахождения координат для процентов

3.3 Круговой график для распределения текущих средств

Первый этап – это создание нового компонента в каталоге Libraries для разработки данного графика в проекте BaiterekLibraries, для этого пишем команду:

ng generate material-costs-pie-chart

Второй этап – это создание дизайна для нашего графика по визуализации данных по затраченным деньгам на материалы. Первым делом разработаем восемь отдельных частей нашего графика с выведенными линиями. Для этого напишем фрагмент кода как показано на рисунке 3.9.

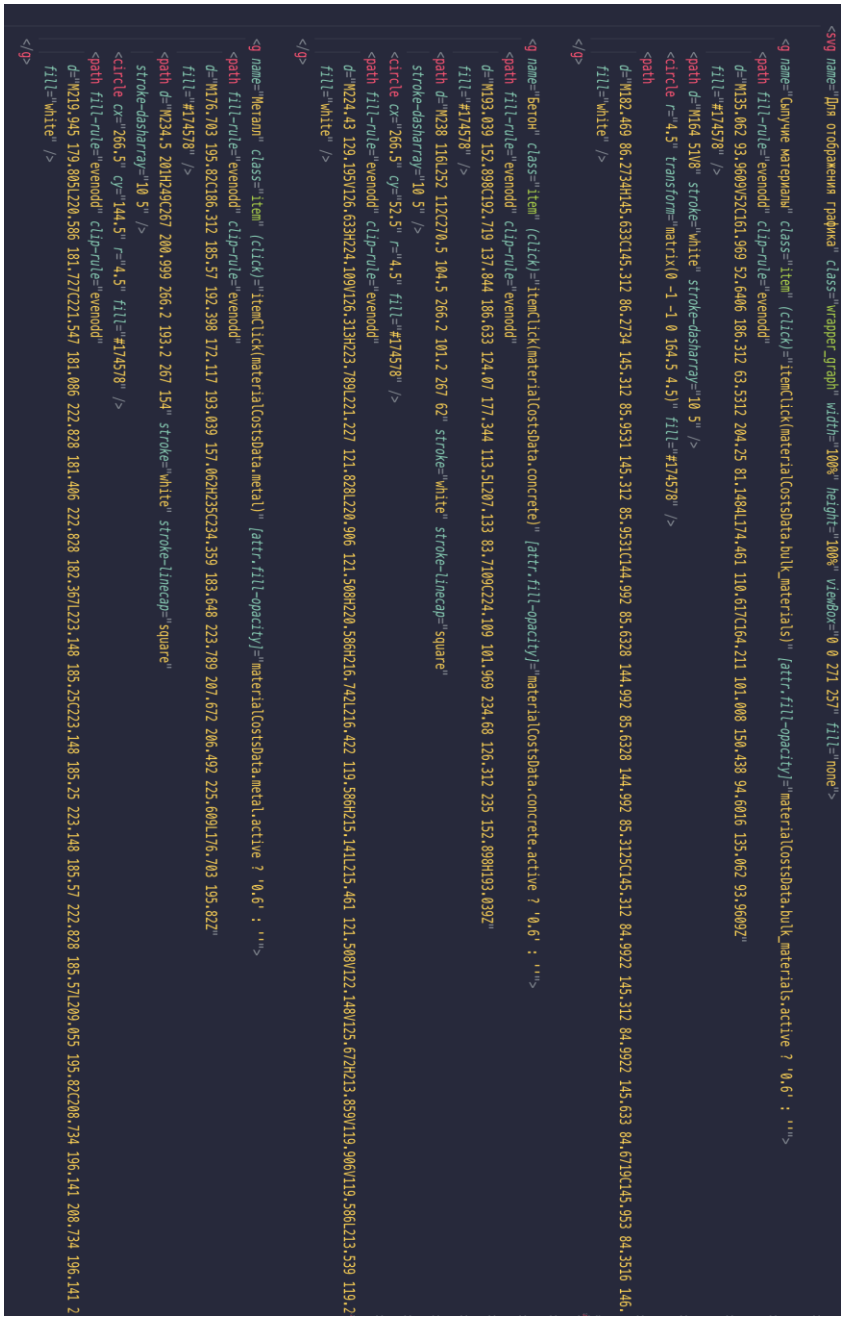


Рисунок 3.9 html код для дизайна графика

Третим этапом создаим текстовые и числовые отображения информации полученных данных. Для этого напишем следующий фрагмент кода в соответствии с рисунком 3.10.

```
<svg name="Для отображения Текстов в Графике" width="100%" height="100%" viewBox="0 0 271 257" fill="none">
  <g name="Сыпучие материалы" class="item">
    <text class="item-total" x="170" y="15">{{materialCostsData.bulk_materials.totalValue  | numberWithSpaces}}</text>
    <text name="Сыпучие материалы" class="item-name" x="170" y="28">Сыпучие материалы</text>

    <rect class="item-info" x="170" y="33" rx="2" ry="2" width="62px" height="10px"></rect>
    <text class="item-info_inner" x="174" y="40" fill="white">{{materialCostsData.bulk_materials.firstValue  | numberWithSpaces}}</text>
    <text class="item-info_inner" x="200" y="40" fill="white">Цемент</text>

    <rect class="item-info" x="170" y="45" rx="2" ry="2" width="62px" height="10px"></rect>
    <text class="item-info_inner" x="174" y="52" fill="white">{{materialCostsData.bulk_materials.secondValue  | numberWithSpaces}}</text>
    <text class="item-info_inner" x="200" y="52" fill="white">Песок</text>

    <rect class="item-info" x="170" y="57" rx="2" ry="2" width="62px" height="10px"></rect>
    <text class="item-info_inner" x="174" y="64" fill="white">{{materialCostsData.bulk_materials.thirdValue  | numberWithSpaces}}</text>
    <text class="item-info_inner" x="200" y="64" fill="white">Прочее</text>
  </g>

  <g name="Бетон" class="item">
    <text class="item-total" x="250" y="52">{{materialCostsData.concrete.totalValue  | numberWithSpaces}}</text>
    <text name="Бетон" class="item-name" x="250" y="65">Бетон</text>

    <rect class="item-info" x="250" y="70" rx="2" ry="2" width="120px" height="10px"></rect>
    <text class="item-info_inner" x="254" y="77.5" fill="white">{{materialCostsData.concrete.firstValue  | numberWithSpaces}}</text>
    <text class="item-info_inner" x="290" y="77.5" fill="white">Тяжелый класс B25</text>

    <rect class="item-info" x="250" y="82" rx="2" ry="2" width="120px" height="10px"></rect>
    <text class="item-info_inner" x="254" y="89.5" fill="white">{{materialCostsData.concrete.secondValue  | numberWithSpaces}}</text>
    <text class="item-info_inner" x="290" y="89.5" fill="white">Тяжелый класс B15</text>

    <rect class="item-info" x="250" y="94" rx="2" ry="2" width="120px" height="10px"></rect>
    <text class="item-info_inner" x="254" y="101.5" fill="white">{{materialCostsData.concrete.thirdValue  | numberWithSpaces}}</text>
    <text class="item-info_inner" x="290" y="101.5" fill="white">Прочее</text>
  </g>
```

Рисунок 3.10 html код для отображения текстов в графике

Четвертый этап – это создание массива данных который будет отображаться на данном графике. Созданный массив опишем в следующем классе MaterialCostsData в соответствии с рисунком 3.11.

```
export interface MaterialCostsData {
  bulk_materials: ItemData;
  concrete: ItemData;
  metal: ItemData;
  masonry_material: ItemData;
  window_and_door: ItemData;
  decoration_material: ItemData;
  plumbing_and_electric: ItemData;
  other: ItemData;
}
```

Рисунок 3.11 Класс MaterialCostsData

Для отображения числовых данных всех частей этого графика, каждый из объектов в классе MaterialCostsData описан следующим классом ItemData как показано на рисунке 3.12.

```
You, 13 days ago | 1 author (You)
export interface ItemData {
  id: number;
  active: boolean;
  totalValue: number;
  firstValue: number;
  secondValue: number;
  thirdValue: number;
}
```

Рисунок 3.12 Класс ItemData

Пятый этап – это создание функции, которая позволит выделять каждую из частей данного графика. Для этого напишем следующий фрагмент кода в соответствии с рисунком 3.13.

```
itemClick(item: ItemData) {
  if (this.itemSelected) {
    this.itemSelected.active = false;
  }
  item.active = true;
  this.itemSelected = item;
}
```

Рисунок 3.13 Функция выделения графика

3.4 Круговой график с картой для визуализации данных

Первым этапом для данного графика будет создание нового компонента в каталоге Libraries нашего проекта. Для этого напишем команду:

ng generate pie-and-map-chart

Второй этап – это создание дизайна для данного графика. В первую очередь при разработке дизайна создадим карту находящуюся по середине данного графика. На данной карте будут отображены названия каждой из областей или города Республики Казахстан. Данный фрагмент html кода приведен в соответствии с рисунком 3.14.

```
<svg width="100%" height="100%" viewBox="0 0 4086 2575" fill="none" *ngIf="!loadingGraph">      You, 18 days ago

  <g class="circle-outside" (click)="outsideClick()">
    <path d="M0 0 L 4086 0 L 4086 2575 L 0 2575 Z" fill="transparent" />
  </g>

  <!-- Карта -->
  <svg width="50%" height="50%" viewBox="0 0 1943 1067" fill="none" x="850" y="600">

    <g class="map-outside" (click)="outsideClicked()">
      <path d="M0 500 L180 0 L1780 0 L 1943 500 L1840 1067 L110 1067 Z" fill="transparent" />
    </g>
    <g class="map" name="ЗКО" (click)="regionCilck(regions[6])">
      <path class="map-area" d="M382.211 425.969L379.951 424.469L377.511 420.009L378.031 412.169L375.921 404.549L
        [attr.fill]="regions[6].color" stroke-width="0.899968" stroke-miterlimit="22.9256" />
    </g>

    <g class="map" name="Туркестанская" (click)="regionCilck(regions[16])">
      <path class="map-area" d="M1164.14 959.32L1163.15 958.79L1161.25 956.91L1160.99 955.95L1161.55 954.95L1162.
        [attr.fill]="regions[16].color" stroke-width="0.899968" stroke-miterlimit="22.9256" />
    </g>

    <g class="map" name="Павлодарская" (click)="regionCilck(regions[13])">
      <path class="map-area" d="M1556.77 293.623L1551.66 284.043L1546.53 274.444V274.434L1539.74 262.004L1532.94
        [attr.fill]="regions[13].color" stroke-width="0.899968" stroke-miterlimit="22.9256" />
    </g>

    <g class="map" name="СКО" (click)="regionCilck(regions[14])">
      <path class="map-area" d="M1304.38 204.016L1298.98 205.376L1295.52 212.036L1289.7 210.706L1290.44 203.986L1
        [attr.fill]="regions[14].color" stroke-width="0.899968" stroke-miterlimit="22.9256" />
    </g>

    <g class="map" name="Мангистауская" (click)="regionCilck(regions[11])">
      <path class="map-area" d="M201.898 790.026L202.218 790.875L202.028 791.536L202.158 791.935L202.658 792.105V
        [attr.fill]="regions[11].color" stroke-width="0.899968" stroke-miterlimit="22.9256" />
    </g>

    <g class="map" name="Кызылординская" (click)="regionCilck(regions[10])">
      <path class="map-area" d="M1034.53 852.353L1034.51 848.063L1023.36 839.714L1018.84 834.224L1018.33 832.234L
        [attr.fill]="regions[10].color" stroke-width="0.899968" stroke-miterlimit="22.9256" />
    </g>
```

Рисунок 3.14 html код карты Республики Казахстан

Следующим действием разработаем дизайн месяцев вокруг нашего графика для обоих годов. На следующем приведенном фрагменте кода будут

прорисованы данные по месяцам в числовом виде, а также линии плана и факта от месяцев к областям либо городам как показано на рисунке 3.15.

```
<!-- Январь верхний полукруг -->
<g *ngIf="checkFact">
  <g *ngFor="let fact of data[0].months[0].lineFacts">
    <path name="факт" [attr.d]="fact.line" fill="transparent" *ngIf="fact.visible"
      stroke="#C7A9ED" stroke-width="3.99986" [attr.stroke-opacity]="fact.active ? '' : '0.101961'" />
  </g>
</g>
<g *ngIf="checkPlan">
  <g *ngFor="let plan of data[0].months[0].linePlans">
    <path name="план" [attr.d]="plan.line" fill="transparent" *ngIf="plan.visible"
      stroke="#C7A9ED" stroke-width="3.99986" stroke-dasharray="27 13.5" [attr.stroke-opacity]="plan.active ? '' : '0.101961'" />
  </g>
</g>
<g name="Январь" class="month" (click)="monthClick(data[0].months[0])">
  <path d="M779.959 1176.74L810.068 1016.77C812.127 1005.82 822.667 998.625 833.607 1000.68C844.556 1002.74 851.756 1013.28 849.696 1024
    [attr.fill]="data[0].months[0].active ? '#C7A9ED' : '#443e59'" />
  <path d="M798.538 1078.04C777.499 1048.18 718.991 973.826 666.473 967.046C620.414 967.136 565.286 967.046 565.286 967.046"
    stroke="#C7A9ED" stroke-width="3.99986" [attr.stroke-opacity]="data[0].months[0].active ? '' : '0.301961'" />
  <path d="M792.178 1111.83C756.509 1093.13 757.989 1048.81 705.401 1047.89C573.416 1048.16 565.286 1047.89 565.286 1047.89"
    stroke="#C7A9ED" stroke-width="3.99986" stroke-dasharray="27 13.5" [attr.stroke-opacity]="data[0].months[0].active ? '' : '0.301961'
  <path d="M814.797 1150.57L814.137 1154.76L805.007 1147.92C804.177 1149.2 803.177 1150.11 802.017 1150.65C800.837 1151.19 799.518 1151.
    fill="#0C1019" />
  <text name="план" [ngClass]="data[0].months[0].active ? 'upNumberText' : 'downNumberText'" x="535" y="1070"
    text-anchor="end">{{summFormat(data[0].months[0].plan)}}</text>
  <text name="факт" [ngClass]="data[0].months[0].active ? 'upNumberText' : 'downNumberText'" x="535" y="990"
    text-anchor="end">{{summFormat(data[0].months[0].fact)}}</text>

</g>
<!-- Январь верхний полукруг -->
```

Рисунок 3.15 html код месяцев и линий

Последующим действием будет разработка общих числовых данных в правой части данного графика и отображения годов в левой части. Данный фрагмент кода приведен в соответствии с рисунком 3.16.

```
<!-- Линия для отображения года слева -->
<g class="arrow" name="Year UP" (click)="arrowClick('up')">
  <path d="M754.799 1454.03C751.11 1425.79 747.41 1397.56 743.72 1369.32C740.46 1344.39 725.11 1323.93 701.881 1323.93H7.16613C4.80622 1
    fill="#808080" />
  <text class="yearUp" x="150" y="1220">{{endYear}}</text>
  <path d="M68.0838 1145L40.2349 1197.36H95.9328L68.0838 1145Z" fill="#73C2FB" />
</g>
<g class="arrow" name="Year Down" (click)="arrowClick('down')">
  <path d="M754.799 1132.95C751.11 1161.19 747.41 1189.42 743.72 1217.66C740.46 1242.59 725.11 1263.05 701.881 1263.05H7.16613C4.80622 1
    fill="#73C2FB" />
  <text class="yearDown" x="150" y="1460">{{startYear}}</text>
  <path d="M68.0838 1448.13L40.2349 1395.77H95.9328L68.0838 1448.13Z" fill="#808080" />
</g>
<!-- Линия для отображения года слева -->

<!-- Линия для отображения общего плана и факта с права -->
<path d="M2950.05 1455.77C2953.74 1427.53 2957.44 1399.3 2961.13 1371.06C2964.39 1346.13 2979.74 1325.67 3002.97 1325.67H4078.78C4081.14
  fill="#808080" />
<text class="totalValue" x="4000" y="1134.69" text-anchor="end">{{summFormat(totalUp.plan, 'year')}} <tspan class="money">млрд. тг
  </tspan>
  план </text>
<text class="totalValue" x="4000" y="1234.69" text-anchor="end">{{summFormat(totalUp.fact, 'year')}} <tspan class="money">млрд. тг
  </tspan>
  факт </text>
<path d="M2950.05 1134.69C2953.74 1162.93 2957.44 1191.16 2961.13 1219.4C2964.39 1244.33 2979.74 1264.79 3002.97 1264.79H4078.99C4081.35
  fill="#73C2FB" />
<text class="totalValue" x="4000" y="1434.69" text-anchor="end">{{summFormat(totalDown.plan, 'year')}} <tspan class="money">млрд. тг
  </tspan>
  план
</text>
<text class="totalValue" x="4000" y="1534.69" text-anchor="end">{{summFormat(totalDown.fact, 'year')}} <tspan class="money">млрд. тг
  </tspan>
  факт
</text>
<!-- Линия для отображения общего плана и факта с права -->
```

Рисунок 3.16 html код для числового отображения справа

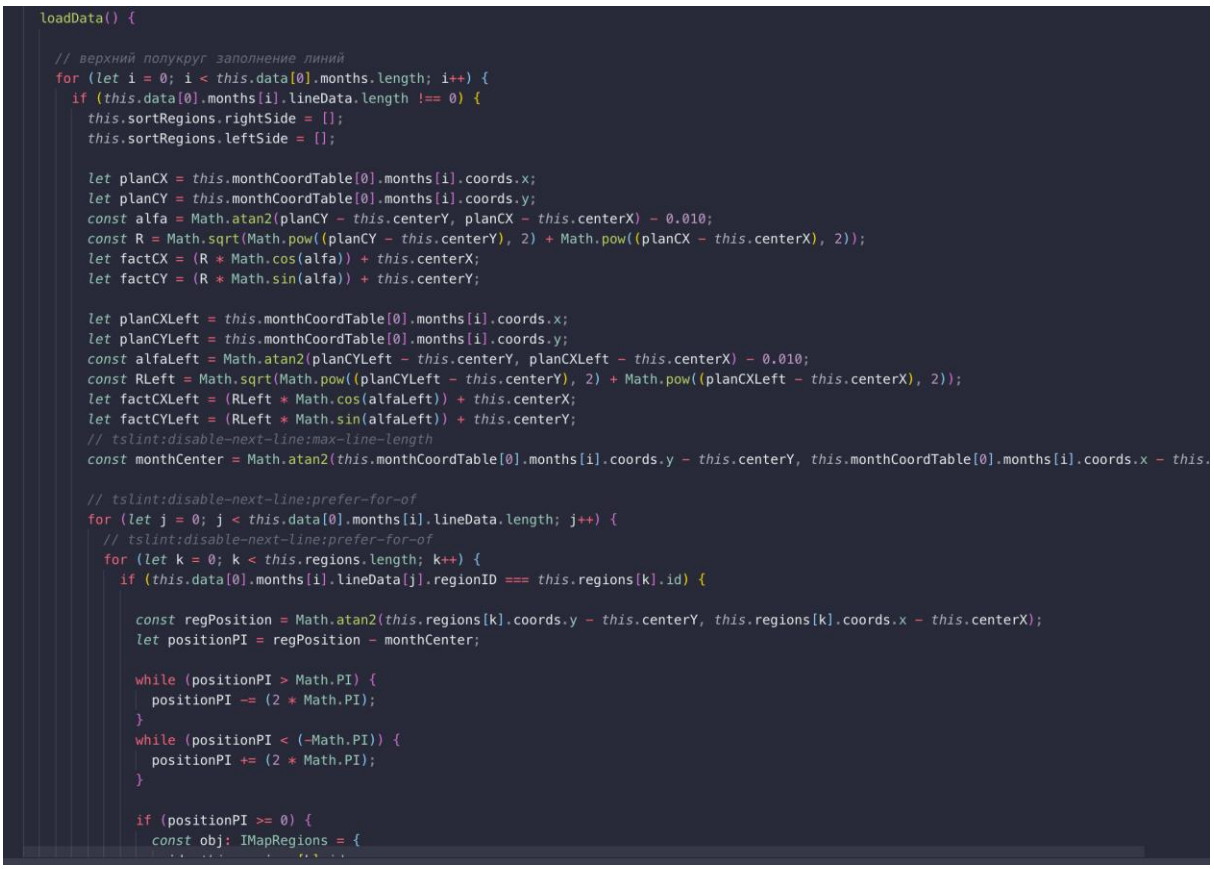
Третий этап – это создание функциональности для данного графика. Первым делом создадим массив с данными для месяцев в обоих годах. Фрагмент кода приведен в соответствии с рисунком 3.17.

```
monthCoordTable: IMonthCoordTable[] = [
  {
    bias: 'up', months: [
      { id: 1, name: 'Январь', coords: { x: 824.5, y: 1102.6 } },
      { id: 2, name: 'Февраль', coords: { x: 902.7438135000315, y: 854.7130519559116 } },
      { id: 3, name: 'Март', coords: { x: 1039.8834287158506, y: 633.8901380631337 } },
      { id: 4, name: 'Апрель', coords: { x: 1227.3921685982696, y: 453.8609575691165 } },
      { id: 5, name: 'Май', coords: { x: 1453.6116478852227, y: 325.8188529598126 } },
      { id: 6, name: 'Июнь', coords: { x: 1714.8064904721273, y: 256.3212660688346 } },
      { id: 7, name: 'Июль', coords: { x: 1985.089046681639, y: 256.30760887648694 } },
      { id: 8, name: 'Август', coords: { x: 2246.2909112656357, y: 325.77879941983633 } },
      { id: 9, name: 'Сентябрь', coords: { x: 2472.523329156987, y: 453.79804194482915 } },
      { id: 10, name: 'Октябрь', coords: { x: 2660.050261584103, y: 633.8082721433267 } },
      { id: 11, name: 'Ноябрь', coords: { x: 2797.212192160602, y: 854.6173257669673 } },
      { id: 12, name: 'Декабрь', coords: { x: 2875.481056376252, y: 1102.4963653327045 } },
    ]
  },
  {
    bias: 'down', months: [
      { id: 1, name: 'Январь', coords: { x: 2879.307092200678, y: 1455.2147691502917 } },
      { id: 2, name: 'Февраль', coords: { x: 2806.4336393729973, y: 1704.7332919789824 } },
      { id: 3, name: 'Март', coords: { x: 2674.093775260154, y: 1928.4657074404759 } },
      { id: 4, name: 'Апрель', coords: { x: 2490.51575163497, y: 2112.5014175717206 } },
      { id: 5, name: 'Май', coords: { x: 2267.113560860751, y: 2245.3979732792322 } },
      { id: 6, name: 'Июнь', coords: { x: 2007.480126712559, y: 2320.518810934859 } },
      { id: 7, name: 'Июль', coords: { x: 1737.2608715599895, y: 2326.3681290538543 } },
      { id: 8, name: 'Август', coords: { x: 1474.6199455513483, y: 2262.552736216462 } },
      { id: 9, name: 'Сентябрь', coords: { x: 1245.6762048585108, y: 2139.4479151930313 } },
      { id: 10, name: 'Октябрь', coords: { x: 1047.6110019855855, y: 1955.538057413266 } },
      { id: 11, name: 'Ноябрь', coords: { x: 912.4089513907827, y: 1745.732416630492 } },
      { id: 12, name: 'Декабрь', coords: { x: 828.8063822828049, y: 1499.6010618619744 } },
    ]
  }
];
```

Рисунок 3.17 Массив данных для месяцев

Данный массив представляет собой информацию, в которой записаны названия всех месяцев и их ID. Так же в данном массиве записан координат по оси x и y, который позволяет запомнить начало исходящей линии от месяца к области или городу.

Следующим этапом будет заполнение данных для линий от месяцев к областям. Фрагмент кода приведен в соответствии с рисунком 3.18.



```
loadData() {

  // верхний полукруг заполнение линий
  for (let i = 0; i < this.data[0].months.length; i++) {
    if (this.data[0].months[i].lineData.length !== 0) {
      this.sortRegions.rightSide = [];
      this.sortRegions.leftSide = [];

      let planCX = this.monthCoordTable[0].months[i].coords.x;
      let planCY = this.monthCoordTable[0].months[i].coords.y;
      const alfa = Math.atan2(planCY - this.centerY, planCX - this.centerX) - 0.010;
      const R = Math.sqrt(Math.pow((planCY - this.centerY), 2) + Math.pow((planCX - this.centerX), 2));
      let factCX = (R * Math.cos(alfa)) + this.centerX;
      let factCY = (R * Math.sin(alfa)) + this.centerY;

      let planCXLeft = this.monthCoordTable[0].months[i].coords.x;
      let planCYLeft = this.monthCoordTable[0].months[i].coords.y;
      const alfaLeft = Math.atan2(planCYLeft - this.centerY, planCXLeft - this.centerX) - 0.010;
      const RLeft = Math.sqrt(Math.pow((planCYLeft - this.centerY), 2) + Math.pow((planCXLeft - this.centerX), 2));
      let factCXLeft = (RLeft * Math.cos(alfaLeft)) + this.centerX;
      let factCYLeft = (RLeft * Math.sin(alfaLeft)) + this.centerY;
      // tslint:disable-next-line:max-line-length
      const monthCenter = Math.atan2(this.monthCoordTable[0].months[i].coords.y - this.centerY, this.monthCoordTable[0].months[i].coords.x - this.

      // tslint:disable-next-line:prefer-for-of
      for (let j = 0; j < this.data[0].months[i].lineData.length; j++) {
        // tslint:disable-next-line:prefer-for-of
        for (let k = 0; k < this.regions.length; k++) {
          if (this.data[0].months[i].lineData[j].regionID === this.regions[k].id) {

            const regPosition = Math.atan2(this.regions[k].coords.y - this.centerY, this.regions[k].coords.x - this.centerX);
            let positionPI = regPosition - monthCenter;

            while (positionPI > Math.PI) {
              positionPI -= (2 * Math.PI);
            }
            while (positionPI < (-Math.PI)) {
              positionPI += (2 * Math.PI);
            }

            if (positionPI >= 0) {
              const obj: IMapRegions = {
```

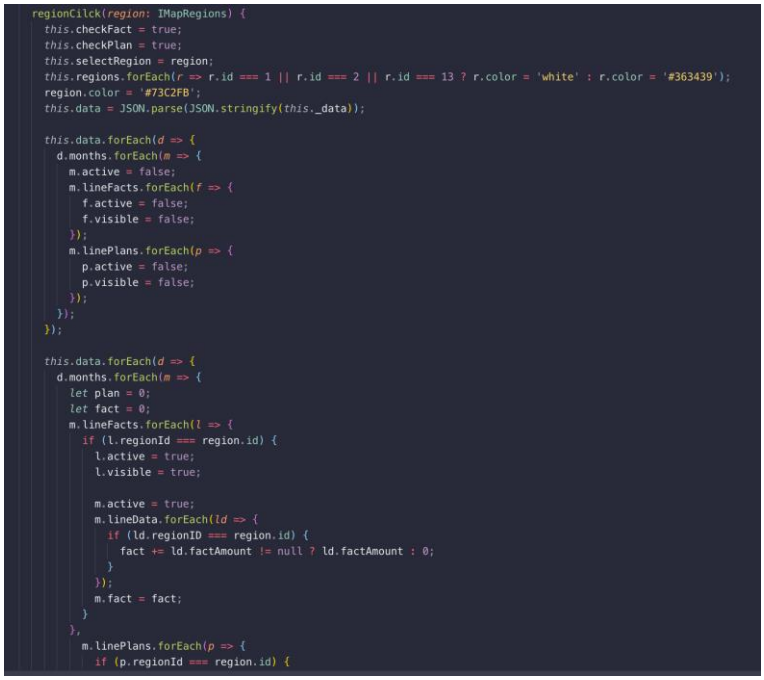
Рисунок 3.18 Функция заполнения данных для линий

Для того что бы лини от месяцев к областям были прорисованы согласно дизайну, нужно написать следующий фрагмент кода в соответствии с рисунком 3.19.

```
lineCalculate(monthCoord: ICoord, regionCoord: ICoord) {
  const angle = -(Math.atan2(monthCoord.y - this.centerY, monthCoord.x - this.centerX));
  const CCoord: ICCoord = { firstCoord: { x: 0, y: 0 }, secondCoord: { x: 0, y: 0 } };
  const newMonthCoord: ICoord = { x: 0, y: 0 };
  const newRegionCoord: ICoord = { x: 0, y: 0 };
  const distance = Math.sqrt(Math.pow((regionCoord.x - monthCoord.x), 2) + Math.pow((regionCoord.y - monthCoord.y), 2));

  newMonthCoord.x = (monthCoord.x - this.centerX) * Math.cos(angle) - (monthCoord.y - this.centerY) * Math.sin(angle);
  newMonthCoord.y = (monthCoord.x - this.centerX) * Math.sin(angle) + (monthCoord.y - this.centerY) * Math.cos(angle);

  newRegionCoord.x = (regionCoord.x - this.centerX) * Math.cos(angle) - (regionCoord.y - this.centerY) * Math.sin(angle);
  newRegionCoord.y = (regionCoord.x - this.centerX) * Math.sin(angle) + (regionCoord.y - this.centerY) * Math.cos(angle);

  CCoord.firstCoord.x = newMonthCoord.x - (distance / 2);
  CCoord.firstCoord.y = newMonthCoord.y;

  CCoord.secondCoord.x = newRegionCoord.x + (distance / 8);
  CCoord.secondCoord.y = newRegionCoord.y;

  const newFirstCX = (CCoord.firstCoord.x * Math.cos(-angle) - CCoord.firstCoord.y * Math.sin(-angle) + this.centerX);
  const newFirstCY = (CCoord.firstCoord.x * Math.sin(-angle) + CCoord.firstCoord.y * Math.cos(-angle) + this.centerY);

  CCoord.firstCoord.x = newFirstCX;
  CCoord.firstCoord.y = newFirstCY;

  const newSecondCX = (CCoord.secondCoord.x * Math.cos(-angle) - CCoord.secondCoord.y * Math.sin(-angle) + this.centerX);
  const newSecondCY = (CCoord.secondCoord.x * Math.sin(-angle) + CCoord.secondCoord.y * Math.cos(-angle) + this.centerY);

  CCoord.secondCoord.x = newSecondCX;
  CCoord.secondCoord.y = newSecondCY;

  // tslint:disable-next-line:max-line-length
  return `M${monthCoord.x} ${monthCoord.y} C${CCoord.firstCoord.x} ${CCoord.firstCoord.y}, ${CCoord.secondCoord.x} ${CCoord.secondCoord.y}, ${regior

}
```

Рисунок 3.19 Функция для изгибания линий

Далее следующий этап – это разработка функциональности при нажатии на регион либо какой-либо месяц. При нажатии на определенный регион видимыми остаются только линии, прорисованные к выбранному региону. Фрагмент кода данной функции приведен согласно рисунку 3.20.



```
regionClick(region: IMapRegions) {
  this.checkFact = true;
  this.checkPlan = true;
  this.selectRegion = region;
  this.regions.forEach(r => r.id === 1 || r.id === 2 || r.id === 13 ? r.color = 'white' : r.color = '#363439');
  region.color = '#73C2FB';
  this.data = JSON.parse(JSON.stringify(this._data));

  this.data.forEach(d => {
    d.months.forEach(m => {
      m.active = false;
      m.lineFacts.forEach(f => {
        f.active = false;
        f.visible = false;
      });
      m.linePlans.forEach(p => {
        p.active = false;
        p.visible = false;
      });
    });
  });

  this.data.forEach(d => {
    d.months.forEach(m => {
      let plan = 0;
      let fact = 0;
      m.lineFacts.forEach(l => {
        if (l.regionId === region.id) {
          l.active = true;
          l.visible = true;

          m.active = true;
          m.lineData.forEach(ld => {
            if (ld.regionID === region.id) {
              fact += ld.factAmount != null ? ld.factAmount : 0;
            }
          });
          m.fact = fact;
        }
      },
        m.linePlans.forEach(p => {
          if (p.regionId === region.id) {
```

Рисунок 3.20 Функция нажатия на регион

При нажатии на определенный месяц видимыми остаются только линии, который были прорисованы от выбранного месяца к остальным областям. Фрагмент кода приведен согласно рисунку 3.21.

```
monthClick(month: IMonth) {
  this.checkFact = true;
  this.checkPlan = true;
  this.selectRegion = null;
  this.regions.forEach(r => r.id === 1 || r.id === 2 || r.id === 13 ? r.color = 'white' : r.color = '#363439');
  this.data[0].months.forEach(m => {
    m.active = false,
      m.lineFacts.forEach(l => {
        l.active = false;
        l.visible = false;
      });
    m.linePlans.forEach(p => {
      p.active = false;
      p.visible = false;
    });
  });
  this.data[1].months.forEach(m => {
    m.active = false,
      m.lineFacts.forEach(l => {
        l.active = false;
        l.visible = false;
      });
    m.linePlans.forEach(p => {
      p.active = false;
      p.visible = false;
    });
  });
  month.active = true;
  month.lineFacts.forEach(f => {
    f.active = true;
    f.visible = true;
  });
  month.linePlans.forEach(p => {
    p.active = true;
    p.visible = true;
  });
  // this.setState();
}
```

Рисунок 3.21 Функция нажатия на месяц

3.5 Карта для отображения аналитических данных

Первый этап – это создание компонента в каталоге Libraries, проекта BaiterekLibrary. Для создание нужного нам компонента, находясь в папке Libraries напишем следующую команду:

ng generate map

Вторым этапом по разработке компонента для визуализации аналитических данных будет создание дизайна. Данная карта отображает все области и города Республики Казахстан. Каждая из областей либо городов имеет возможность выделения, для осуществления данной возможности мы прорисуем каждую область по отдельности. Города на карта должны отображаться кругами в своем месторасположении и также должны выделяться. Фрагмент кода для данного дизайна карты Республики Казахстан представлен в соответствии с рисунком 3.22.

```
<g class="map" name="ВКО" (click)="regionCilck(regions[15])">
  <path class="map-area"
    d="M1940.4 487.716L1939.39 482.606L1938.63 481.096
       L1936.06 479.316L1934.75 478.736L1933.33 478.436L1929.32 478.826L1928.09 478.696
       L1927.44 478.416L1926.88 477.966L1926.5 477.266V476.237L1926.3 475.656L1925.74 475.057L1922.55 472.507
       L1921.58 471.457L1921.2 470.497L1921.7 469.177L1922.41 468.397L1922.54 467.707L1921.22 466.647L1918.79 465.447L1917.
       L1916.86 463.357L1916.85 461.677L1917.22 460.227L1917.34 459.027L1916.28 458.077L1912.96 456.017L1911.87 455.737L190
       L1907.55 455.507L1906.62 454.377L1906.25 453.377L1906.43 452.297L1907.15 450.947L1908.85 448.488L1909.83 447.708L191
       L1914 446.408L1914.97 445.418L1913.8 444.198L1913.29 443.388L1913.16 442.488L1913.41 441.768L1913.84 440.938L1914.14
       L1875.44 456.287L1875.31 455.657L1873.37 455.347L1872.46 455.417L1871.52 455.847L1869.95 457.057L1869.08 457.397L186
    [attr.fill]="regions[15].color" [attr.fill-opacity]="regions[15].active ? '0.7' : ''" stroke-width="2.35992" stroke-miterl
  <text class="map-city" x="1700" y="530" fill="white" text-anchor="middle">{{regions[15].title}}</text>
  <text class="map-city value" x="1700" y="500" fill="white" text-anchor="middle">{{regions[15].value}}</text>
</g>

<g name="Алматы" (click)="regionCilck(regions[1])">
  <circle class="map-circle" cx="1460" cy="880" r="15" fill="transparent" [attr.stroke]="regions[1].color"
  [attr.stroke-opacity]="regions[1].active ? '0.5' : ''" stroke-width="4"></circle>
  <circle class="map-circle" cx="1460" cy="880" r="9" [attr.fill]="regions[1].color" [attr.fill-opacity]="regions[1].active ?
  <text class="map-city" x="1550" y="890" fill="white" text-anchor="middle">{{regions[1].title}}</text>
  <text class="map-city value" x="1550" y="860" fill="white" text-anchor="middle">{{regions[1].value}}</text>
</g>

<g name="Астана" (click)="regionCilck(regions[0])">
  <circle class="map-circle" cx="1130" cy="370" r="15" fill="transparent" [attr.stroke]="regions[0].color"
  [attr.stroke-opacity]="regions[0].active ? '0.5' : ''" stroke-width="4"></circle>
  <circle class="map-circle" cx="1130" cy="370" r="9" [attr.fill]="regions[0].color" [attr.fill-opacity]="regions[0].active ?
  <text class="map-city" x="1250" y="380" fill="white" text-anchor="middle">{{regions[0].title}}</text>
  <text class="map-city value" x="1250" y="350" fill="white" text-anchor="middle">{{regions[0].value}}</text>
</g>

<g name="Шымкент" (click)="regionCilck(regions[12])">
  <circle class="map-circle" cx="1080" cy="960" r="15" fill="transparent" [attr.stroke]="regions[12].color"
  [attr.stroke-opacity]="regions[12].active ? '0.5' : ''" stroke-width="4"></circle>
  <circle class="map-circle" cx="1080" cy="960" r="9" [attr.fill]="regions[12].color" [attr.fill-opacity]="regions[12].active
  <text class="map-city" x="1190" y="970" fill="white" text-anchor="middle">{{regions[12].title}}</text>
  <text class="map-city value" x="1190" y="940" fill="white" text-anchor="middle">{{regions[12].value}}</text>
</g>
</svg>
```

Рисунок 3.22 html код для дизайна карты

Третий этап – это создание массива данных для отображения информации аналитических данных на данной карте в числовом виде.

Созданный массив будет нести информацию о названии и цвете каждой области либо города, так же будет содержать информацию о том какая из областей выделена и какое число будет отображено над данной областью либо городом. Фрагмент кода представлен в соответствии с рисунком 3.23.

```
@Input() regions: IMapRegions[] = [
  { id: 1, title: 'Нур-Султан', color: 'white', value: 0, active: false },
  { id: 2, title: 'Алматы', color: 'white', value: 0, active: false },
  { id: 3, title: 'Акмолинская', color: '#363439', value: 0, active: false },
  { id: 4, title: 'Актюбинская', color: '#363439', value: 0, active: false },
  { id: 5, title: 'Алматинская', color: '#363439', value: 0, active: false },
  { id: 6, title: 'Атырауская', color: '#363439', value: 0, active: false },
  { id: 7, title: 'ЗКО', color: '#363439', value: 0, active: false },
  { id: 8, title: 'Жамбылская', color: '#363439', value: 0, active: false },
  { id: 9, title: 'Карагандинская', color: '#363439', value: 0, active: false },
  { id: 10, title: 'Костанайская', color: '#363439', value: 0, active: false },
  { id: 11, title: 'Кызылординская', color: '#363439', value: 0, active: false },
  { id: 12, title: 'Мангистауская', color: '#363439', value: 0, active: false },
  { id: 13, title: 'Шымкент', color: 'white', value: 0, active: false },
  { id: 14, title: 'Павлодарская', color: '#363439', value: 0, active: false },
  { id: 15, title: 'СКО', color: '#363439', value: 0, active: false },
  { id: 16, title: 'ВКО', color: '#363439', value: 0, active: false },
  { id: 17, title: 'Туркестанская', color: '#363439', value: 0, active: false },
];
```

Рисунок 3.23 Массив с данными для карты Республики Казахстан

Для осуществления функциональности по выделению областей либо городов, а также отправки события в компонент напишем следующий фрагмент кода как показано на рисунке 3.24.

```
@Output() regionClick = new EventEmitter<number>();
@Output() outsideClick = new EventEmitter<any>();

constructor() { }

ngOnInit() {
}

regionCilck(region: IMapRegions) {      You, 15 days ago • chanhes
  this.regions.forEach(r => r.active = false);
  region.active = true;
  this.regionClick.emit(region.id);
}

outsideClicked() {
  this.regions.forEach(r => r.active = false);
  this.outsideClick.emit(true);
}
```

Рисунок 3.24 Функция выделения области на карте

Заключение

В результате выполнения данной работы мы рассмотрели разные возможности SVG и технологию Angular.

Angular library дает возможность создать библиотеки тем самым позволяя решить одну и туже задачу в разных приложения не дублируя код.

SVG позволяет уменьшать размер файла для улучшения производительности не теряя качество изображения.

Использование SVG дает возможность визуализировать разные аналитические данные и многое другое.

Масштабируемая векторная графика позволяет изображению выглядеть четко на мониторе с любым разрешением, при этом иметь очень маленький размер файла и легко поддаваться редактированию и изменениям.

Создали библиотеку для визуализации аналитических данных на базе Angular Framework и опубликовали на сайте npmjs.com.

Актуальность работы исследования состоит в том, что разработчикам для визуализации аналитических данных не придется копировать код либо создавать отдельные компоненты.

Список использованных источников

1) Евгений Попов, Введение в Angular, Начало работы с Фреймворком. //[Электронный ресурс]. URL: https://metanit.com/web/angular2/1.1.php

2) Создание библиотек. //[Электронный ресурс]. URL: https://angular.io/guide/creating-libraries

3) Дмитрий Лебедев, SVG в вебе Практическое руководство, 2017.