**Стоимость:**

Процессор AMD Ryzen Threadripper 1920X – 700 usd – 40 000 rub

Материнка ASUS PRIME X399-A – 300 usd – 17 000 rub

Оперативка Samsung DDR4 2400 DIMM 8Gb х2 – 5 000 x2 – 10 000 rub

Видеокарта Gigabyte GeForce GTX 1070 – 700 usd – 40 000 rub

Сумма: 107 000 rub

Амортизация: 24 месяца

В месяц: 4458 rub

Размещение сервера(<https://www.smartape.ru/colocation>): 8 990 rub

**13 500 rub в месяц**

Приблизительно 300 тайлов в сек. в самом трудном случае

Считаю что для заполнения экрана нужно 40 тайлов.

При 10 тайлах в сек. на пользователя – 450 rub на пользователя пиковой нагрузки

При 5 тайлах в сек. на пользователя – 225 rub на пользователя пиковой нагрузки

Это все в САМОМ ХУДШЕМ СЛУЧАЕ. Т.к самых трудных тайлов всего 16.

Буду считать что будет средняя скорость 900 тайлов в сек.

При 10 тайлах в сек. на пользователя – 150 rub на пользователя пиковой нагрузки

При 5 тайлах в сек. на пользователя – 75 rub на пользователя пиковой нагрузки

Карту можно сделать более информативной раскрашивая дома в цвет, зависящий от цены за квадратный метр, и показать пользователю транспортную доступность раскрашиваю дороги в цвет зависящий от их удаленности от точки отсчета. Градиенты приведены на слайде. Благодаря естественной ассоциации красного как с чем-то более дорогим, а синего как с чем-то дешевым. Также в тепловизоре красным цветом показываются горячие области, а синим наоборот холодные. Поэтому мы можем сопоставить максимальные цены с красным, а минимальные с синим. Тот же принцип применим к дальности.

Для реализации подобного функционала необходимы геоданные и цены. Геоданные нужны для отрисовки и расчета дальности. Цены для того что бы было что визуализировать. Геоданные и цены я брал для области приблизительно ограниченной бетонным кольцом.

В качестве сервера мною был выбран nginx т.к. он очень популярен и используется крупнейшими интернет компаниями. Java была выбрана в качестве языка программирования из-за того что я неплохо владею ей, и её очень часто применяют в сфере написания серверов.

Все запросы сначала попадают на сервер nginx, дальше в зависимости от типа запроса либо nginx отдаёт уже готовую картинку карты, либо перенаправляет запрос на сервер java. Его принимает Netty, дальше он передаётся в PathRouter, который согласно url передаёт запрос нужному handler.

Для отрисовки и поиска по карте необходимо организовать данные на плоскости. Для этого существует несколько способов, но я выбрал квадродерево, как самый простой в реализации. У квадродрева у каждого узла 4 потомка, которые делят прямоугольник родителя на 4 равных части. Разделение узла происходит, когда количество элементов в нем превышает некий порог. Благодаря этому нам никогда не приходится искать перебором в количестве элементов большем чем порог разделения.

**Во время анимации:**

Как видно, после того как количество объектов превысит некий порог, узел делится.(порог 65) При этом объекты хранятся только в конечных узлах, а после разделения распределяются по потомкам.

Из-за плохой стандартизации данных имеются проблемы с маршрутами, остановки не являются частью маршрута. Пришлось делать остановки частью ближайшего к ним ребра. А после соединить остановку с ближайшими вершинами. После этого маршруты общественного транспорта становятся частью дорожного графа.

В начале из-за простоты реализации использовался расчет на основе поиска в глубину, но из-за его плохой производительности я переписал расчет на основе поиска в ширину. Их сравнение приведено на слайде.