САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: “Фронт-энд разработка”

Отчет
Домашняя работа №2

Выполнил: Митурский Богдан Антонович

Группа: K33392

Санкт-Петербург

2023 г.

Цель:

Улучшить доступность ранее реализованной игры. В ходе работы будут проведены следующие действия для улучшения доступности игры:

1. Разделение кода на бандлы для ускорения первого контакта пользователя с страницей и реализация загрузочной страницы
2. Улучшение доступности клиентской части проекта путём размещения на высоко доступном CDN

Программное обеспечение: TypeScript, Node Typescript, Socket.io, React

Ход работы:

1. Для разделения кода на фрагменты будем использовать React Suspense, обернем всё наше приложение в этот компонент:

```
  <Suspense>
      <App />
  </Suspense>
```

2. Т.к. функция в реакте это достаточно новая, никакой индикации того, что контент внутри загрузился у нас нет. Но она необходима нам чтобы реализовать загрузочный экран с красивой анимацией и реализовать логику, при которой загрузочный экран показывается пока контент приложения не загрузился, и анимация не проигралась. Для этого прибегнем к хитрости. Создадим компонент, принимающий в себя функцию onload вызывать которую компонент, будет в момент его удаления со сцены и передадим его в fallback параметр Suspense. Получится что когда переданный в fallback пустой компонент удаляется со сцены у нас срабатывает onload и мы понимаем, что контент внутри прогрузился:

```
<Suspense fallback={<LoadIndicator onLoad={() => setIsLoaded(true)} />}>
      <App />
</Suspense>

const LoadIndicator: React.FC<LoadIndicatorProps> = ({ onLoad }) => {
  useEffect(() => {
    return () => onLoad();
  }, []);
  return <LoadIndicatorStyled />;
};
```

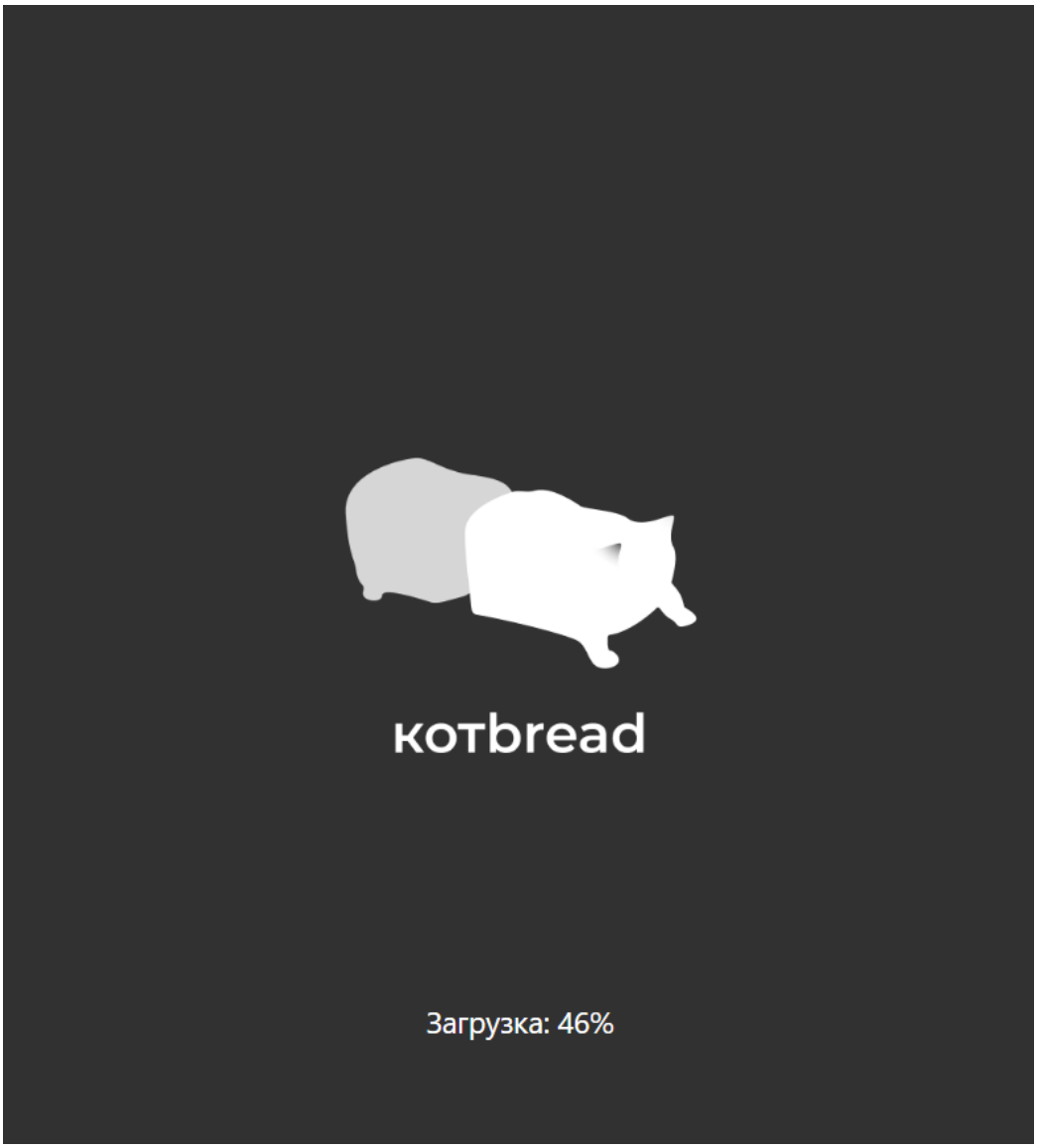
3. Остается поверх всего приложения и Suspense с минимально возможным числом зависимостей реализовать загрузочную страницу:

```
import React, { useEffect, useState } from "react";
import { images } from "../../data/url";
import { loadingSound } from "../../data/sounds/loading";

import styled from "@emotion/styled";
import { keyframes } from "@emotion/react";
import { AppColorScheme } from "./AppSuspense";
import { Background } from "./Background";

type KotbreadPanelProps = {
  onEnd: () => void;
  colorScheme: AppColorScheme;
};

const logoScale = keyframes`
  from {
    scale: 0.8;
    opacity: 0;
  }
  50% {
    scale: 1.1;
  }
  to {
    scale: 1;
    opacity: 1;
  }
`;

const KotBreadLogo = styled.img`
  width: 100%;
  max-width: 260px;
  animation: ${logoScale} 1s ease-out;
`;

const Progress = styled.div<{ colorScheme: AppColorScheme }>`
  position: absolute;
  bottom: 64px;
  width: fit-content;
  text-align: center;
  font-size: 18px;
  color: ${({ colorScheme }) => (colorScheme === "dark" ? "white" : "black")};
  font-family: -apple-system, BlinkMacSystemFont, "Segoe UI", "Roboto",
"Oxygen",
    "Ubuntu", "Cantarell", "Fira Sans", "Droid Sans", "Helvetica Neue",
    sans-serif;
```

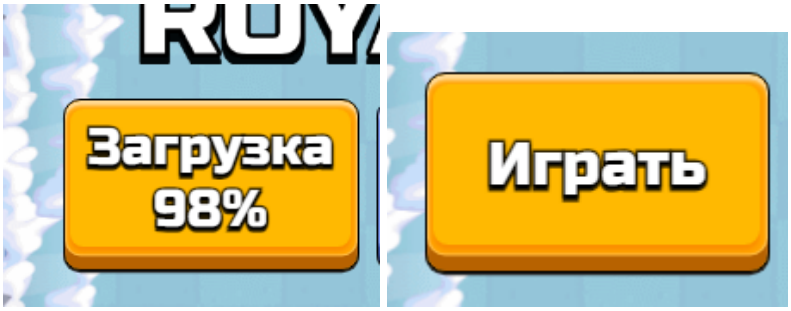
```
`;

const KotbreadPanel: React.FC<KotbreadPanelProps> = ({
  onEnd,
  colorScheme,
}) => {
  const [percent, setPercent] = useState(1);
  const innerString = `Загрузка: ${percent}%`;

  // Проигрывает звуковой эффект при запуске
  useEffect(() => {
    const updateProgress = setInterval(() => {
      setPercent((percent) =>
        Math.floor(
          percent >= 99
            ? 99
            : percent >= 80
            ? percent + ((100 - percent) / 10) * Math.random() + 0.8
            : percent + 15 * Math.random()
        )
      );
    }, Math.random() * 500 + 100);
    loadingSound.once("end", () => {
      onEnd();
    });
    return () => clearInterval(updateProgress);
  }, []);

  return (
    <Background colorScheme={colorScheme}>
      <KotBreadLogo
        src={
          colorScheme === "dark"
            ? images.kotbreadLogo.light
            : images.kotbreadLogo.purple
        }
        alt="Логотип КотBread"
      />
      <Progress colorScheme={colorScheme}>{innerString}</Progress>
    </Background>
  );
};

export default React.memo(KotbreadPanel);
```

4. Как итог. До добавления первичной загрузочной странице пользователю необходимо было сидеть на серой странице браузера загружая все 2мб веса игры. Теперь же пользователь

наблюдает загрузочный экран с анимацией и звуковым эффектом уже после скачивания первых 100кб. К сожалению, на данный момент нет возможности отследить прогресс загрузки контента через React Suspense, поэтому мы используем ненастоящие цифры загрузки, чтобы показывать пользователю прогресс и с большей вероятностью оставить его на странице игры до полной загрузки.



5. Для дополнительного ускорения процесса загрузки вынесем загрузку всех внутриигровых спрайтов уже после попадания в главное меню. Если игрок вдруг нажмет на кнопку “Играть” до того, как все спрайты будут загружены, выведем надпись: “Загрузка” прямо на кнопке и заменим её, когда загрузка спрайтов завершится:




```
<HomeScreenButton
          onClick={handlePlayButtonOneOnOneBattle}
          disabled={!!cloudsPopout && !meta?.gameMode}
          data-tutorial-step={1}
          mode="primary"
          fontSize={24}
          before
        >
          {currentProgress === null ? "Играть" : `Загрузка
${currentProgress}%`}
</HomeScreenButton>

// Промис, который ждет загрузки спрайтов
const waitLoading = () =>
    new Promise<void>((resolve) => {
      const unsubscribe = store.subscribe(() => {
        const progress = store.getState().App.loaderProgress;
        setCurrentProgress(progress === 100 ? null : progress);
        if (progress === 100) {
          unsubscribe();
          resolve();
        }
      });
    });
```

6. Работая над улучшением доступности проекта не стоит забывать про доступность сетей. Скорость доставки проекта пользователю может отличаться от хостинга к хостингу. В рамках данного проекта после тщательного анализа доступных хостинг провайдеров было принято решение использовать Vk MiniApps Hosting. Хостинг статики разработанный специально для создаваемых мини-приложений ВКонтакте. Во-первых, он бесплатен. Во-вторых когда мы делаем игру внутри ВКонтакте, логичным будет разместить её на тех же CDN что и родительское приложение, чтобы пользователь не столкнулся с ситуацией когда ВК работает у него хорошо, а игра не открывается из-за проблем с

доступом из определенного региона или слабой точки wifi сети. Ну и, в-третьих, у самой крупной социальной сети в России своя большая и качественная сеть доставки контента, над которой они постоянно работают. Как итог проект загружается в мир средствами Vk Mini Apps Hosting-а и имеет доступность уровня доступности социальной сети ВКонтакте.

Вывод:

В ходе работы я улучшил доступность игры для конечных пользователей. Уменьшил время первого контакта пользователя с игрой, увеличил удержание во время загрузки, а также улучшил доступность в самых разных регионах России путём размещения приложения на CDN ВКонтакте.