Курс NextJS. Потоковый вывод компонентов

Д. Макаренков, к.т.н.

https://dmpsy.club

NextJS >= ReactJS + NodeJS

Целевая аудитория

Энтузиасты программирования на NextJS, желающие следовать современным принципам разработки full-stack Web-приложений

Магическая формула:

NextJS > = ReactJS (front-end) + NodeJS (back-end)

План работы / Agenda

1	What is streaming in NextJS. Statement of the problem	Потоковый вывод в NextJS. Постановка задачи
2	Streaming the Dashboard page	Потоковый вывод страницы Dashboard
3	Streaming a loading skeleton of the Dashboard page	Вывод заставки (loading skeleton) страницы Dashboard
4	Using a route group	Использование route group
5	Streaming with the <Suspense> wrapper	Вывод компонентов обернутых в <Suspense>
6	Streaming <RevenueChart>	Потоковый вывод <RevenueChart>
7	Streaming <LatestInvoices>	Потоковый вывод <LatestInvoices>

Официальная версия от NextJS: https://nextjs.org/learn/dashboard-app/streaming

План работы / Agenda (2)

8	Streaming similar components in a common “wrapper”	Вывод однородных компонентов в общей “обертке”
9	Streaming <CardWrapper>	Потоковый вывод <CardWrapper>
10	Optimal placement of <Suspense> boundaries	Оптимальное расположение границ <Suspense>
11	Summary & Closing remarks	Итоги и заключительные замечания

A	Launching a project testbed	Запуск тестовой среды
B	Stopping & Removing the testbed	Останов и удаление тестовой среды
C	fetchRevenue() slowdown	“Замедление” fetchRevenue()

Потоковый вывод страниц и компонентов в NextJS. Постановка задачи

В предыдущей главе вы узнали о различных методах рендеринга Next.js. Мы также обсудили, как медленная выборка данных может повлиять на производительность вашего приложения. Давайте рассмотрим, как можно улучшить пользовательский опыт при медленных запросах данных

Потоковая передача данных — это метод передачи данных, позволяющий разбить маршрут на более мелкие «фрагменты» и постепенно передавать их с сервера клиенту по мере готовности

С помощью потоковой передачи вы можете предотвратить блокировку всей страницы медленными запросами данных. Это позволяет пользователю видеть и взаимодействовать с частями страницы, не дожидаясь загрузки всех данных, прежде чем пользователю будет показан какой-либо пользовательский интерфейс.

Два способа реализации потокового вывода в NextJS

Потоковая передача хорошо работает с компонентной моделью React, поскольку каждый компонент можно считать фрагментом.

Есть два способа реализации потоковой передачи в Next.js:

1. На уровне страницы с помощью файла loading.tsx (который создает для вас <Suspense>).
2. На уровне компонента с помощью <Suspense> для более детального управления.

Давайте посмотрим, как это работает.

NB Фрагменты обрабатываются параллельно, что сокращает общее время загрузки.

Потоковый вывод страницы Dashboard

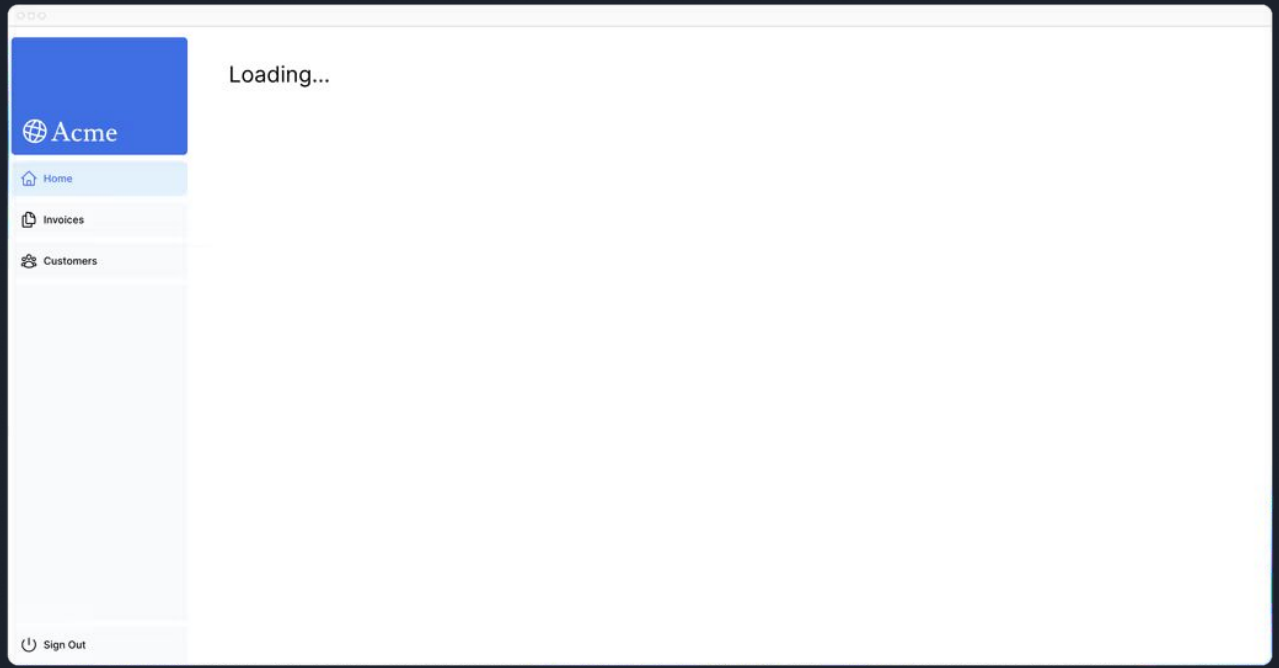
0	Launch the project environment	Склонируйте и запустите проект: Slide 28: Приложение. Запуск рабочей среды проекта

1	Create the loading.tsx file	Создайте файл app/dashboard/loading.tsx:
		export default function Loading() {
		  return Loading... ;
		}

1a	Copy the loading.tsx file	(Или) скопируйте файл loading.tsx: cp loading.tsx.00.JustLoading loading.tsx

Потоковый вывод страницы Dashboard (2)

2	**Open and refresh localhost:3000/dashboard**	**Откройте и обновите страницу http://localhost:3000\dashboard При перезагрузке на мгновение появляется loading.tsx:**



Потоковый вывод страницы Dashboard (3)

Отметим следующее:

1. loading.tsx — это специальный файл Next.js, созданный поверх React Suspense. Он позволяет создать страницу-заставку, которая будет отображаться в качестве замены во время загрузки содержимого страницы.
2. Поскольку <SideNav> является статическим, он отображается немедленно. Пользователь может взаимодействовать с <SideNav> во время загрузки динамического содержимого.
3. Пользователю не нужно ждать завершения загрузки страницы, прежде чем уйти (это называется прерываемой навигацией).

Теперь выведем скелет загрузки (loading skeleton) вместо текста Loading.... в качестве заставки

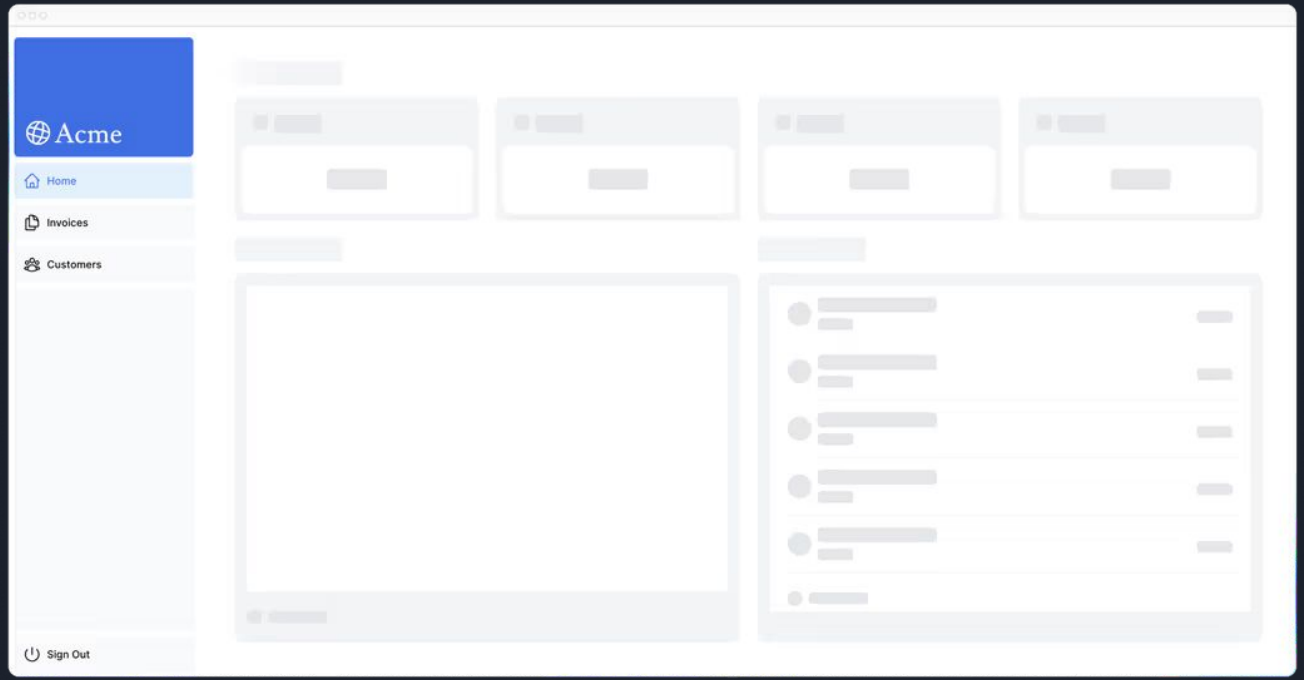
Вывод заставки (loading skeleton) страницы Dashboard

1	**Create the loading.tsx file**	**Отредактируйте файл app/dashboard/loading.tsx:**
		`import DashboardSkeleton from '@/app/ui/skeletons';`
		
		`export default function Loading() {`
		`  return <DashboardSkeleton />;`
		`}`

1a	**Copy the loading.tsx file**	**(Или) скопируйте файл loading.tsx:** `cp loading.tsx.01.DashboardSkeleton loading.tsx`

Вывод заставки (loading skeleton) страницы Dashboard (2)

2	**Open and refresh localhost:3000/dashboard**	**Обновите страницу http://localhost:3000\dashboard При перезагрузке на мгновение появляется заставка-” скелет” страницы loading.tsx:**



Ограничиваем область действия заставки с помощью route group

Проблема:

Поскольку loading.tsx сейчас находится на уровень выше, чем /invoices/page.tsx и /customers/page.tsx в файловой системе, заставка из loading.tsx также применяется и к этим страницам

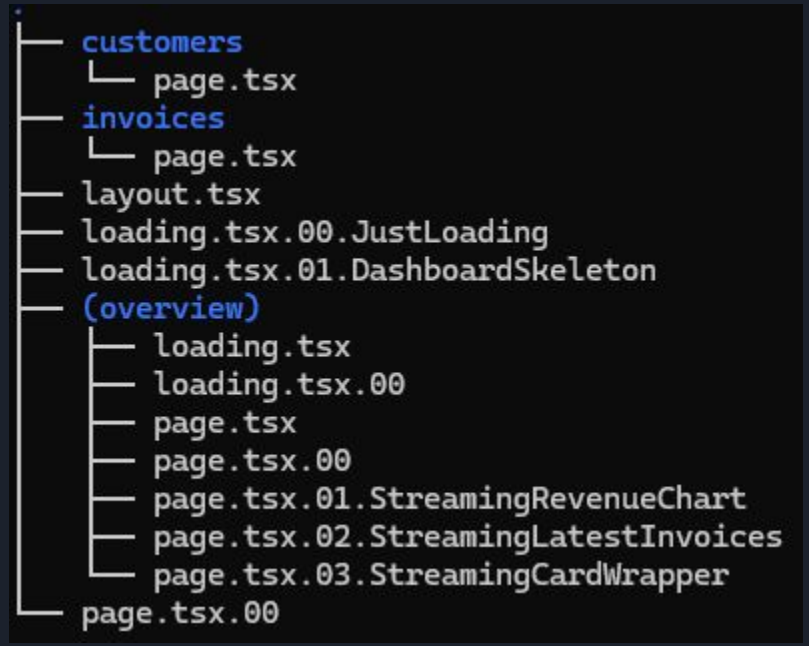
Решение:

Вводим невидимый для маршрутизации в браузере уровень - “группу маршрутов” (route groups), для чего создаем новую папку (overview) внутри папки dashboard и перемещаем туда loading.tsx и page.tsx:

```
cd app/dashboard
rm loading.tsx page.tsx
mv overview '(overview)'
cd '(overview)'
cp loading.tsx.00 loading.tsx
cp page.tsx.00 page.tsx
```

Далее убеждаемся, что заставка применяется теперь только к Dashboard

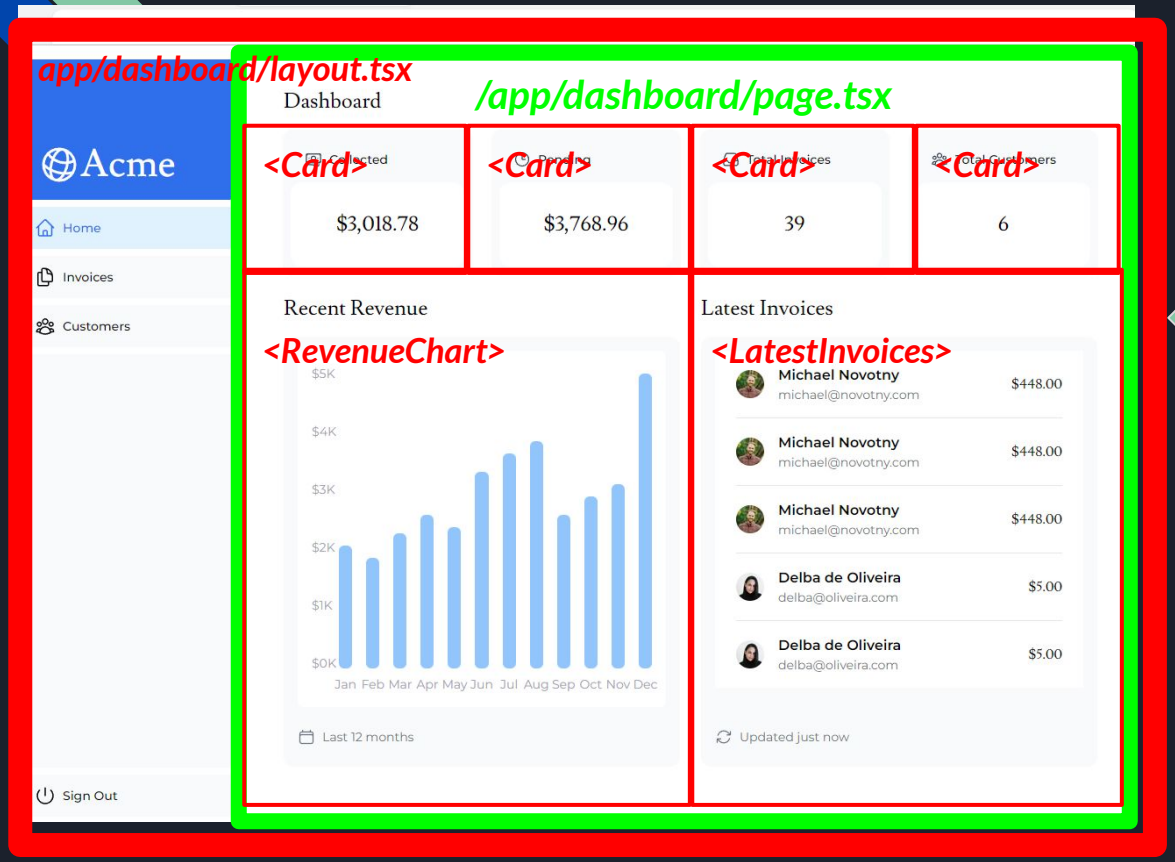
Ограничиваем область действия заставки с помощью route group (2)



1. Группы маршрутов (route groups) позволяют вам организовывать файлы в логические группы, не влияя на структуру пути URL. Если имя папки помещено в скобки (), она не будет включена в путь URL в браузере: /dashboard/(overview)/page.tsx идентично /dashboard

2. Сейчас мы использовали группу маршрутов, чтобы гарантировать, что loading.tsx применяется только к странице (dashboard) обзора панели мониторинга. Однако, группы маршрутов помогают также упорядочить разделы приложения (например, маршруты (marketing) и (shop))

Потоковый вывод компонентов оборачиванием их в <Suspense>



/app/lib/data.ts
fetchRevenue()
fetchLatestInvoices()
fetchCardData()

SQL

nexxtapp-db

Потоковый вывод компонентов оборачиванием их в <Suspense> (2)

Элемент <Suspense> позволяет отложить отрисовку “медленного” компонента до тех пор, пока не будут подгружены его данные: вы просто оборачиваете этот компонент в элемент <Suspense> и передаете ему резервный (fallback) компонент - заставку для отображения во время загрузки данных.

Например, функция fetchRevenue() замедляет сейчас отрисовку всей страницы. Для снятия этой блокировки обернем <RevenueChart> в <Suspense>, а вызов функции fetchRevenue() переместим в <RevenueChart>, в результате чего все компоненты страницы dashboard будут отображаться практически мгновенно, а <RevenueChart> - по получении данных

Потоковый вывод компонента <RevenueChart>

1	Edit the page.tsx file	Отредактируйте файл page.tsx: 1. Удалите все экземпляры fetchRevenue() и его данные из /dashboard/(overview)/page.tsx в компонент <RevenueChart> 2. Импортируйте <Suspense> из React и оберните его вокруг <RevenueChart />. 3. Передайте ему fallback-компонент <RevenueChartSkeleton>.
1a	Copy the page.tsx file	(Или) скопируйте файл page.tsx: cd '(overview)' cp page.tsx.01.StreamingRevenueChart page.tsx
2	Review the page.tsx file	Изучите файл page.tsx cat page.tsx

Потоковый вывод компонента <RevenueChart> (2)

Убираем fetchRevenue() в тело <RevenueChart /> и оборачиваем его в <Suspense> с заставкой <RevenueChartSkeleton>. Вызываем <RevenueChart> без параметров (props)

/app/dashboard/’(overview)’/page.tsx:

```
import { Card } from '@/app/ui/dashboard/cards';
…
import { fetchLatestInvoices, fetchCardData, } from '@/app/lib/data'; //DM fetchRevenue removed
import { Suspense } from 'react';  //DM <Suspense> added
import { RevenueChartSkeleton } from '@/app/ui/skeletons';  //DM <RevenueChartSkeleton added>

export default async function Page() {
  //DM const revenue = await fetchRevenue(); fetchRevenue call moved to <RevenueChart />
  const latestInvoices = await fetchLatestInvoices();
…
 <div className="mt-6 grid grid-cols-1 gap-6 md:grid-cols-4 lg:grid-cols-8">
       <Suspense fallback={<RevenueChartSkeleton />}>
         <RevenueChart />
       </Suspense>
       <LatestInvoices latestInvoices={latestInvoices} />
      </div>
```

Потоковый вывод компонента <RevenueChart> (3)

3	Edit the revenue-chart.tsx file	Отредактируйте файл app/ui/dashboard/revenue-chart.tsx: Обновите компонент <RevenueChart> так, чтобы он извлекал свои данные с fetchRevenue() и удалите его prop
3a	Copy the revenue-chart.tsx file	(Или) скопируйте файл revenue-chart.tsx: cd app/ui/dashboard cp revenue-chart.tsx.01.StreamingRevenueChart revenue-chart.tsx
4	Review the revenue-chart.tsx file	Изучите файл revenue-chart.tsx cat revenue-chart.tsx

Потоковый вывод компонента <RevenueChart> (4)

Корректируем заголовок RevenueChart (убираем props), импортируем и вызываем fetchRevenue() в теле функции RevenueChart() {...}

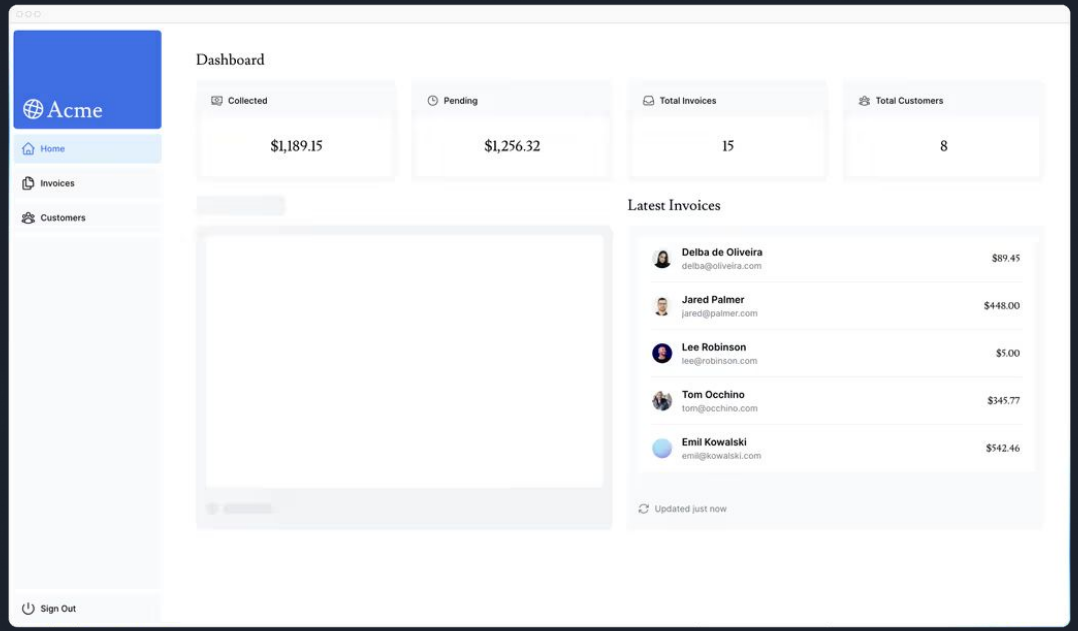
/app/ui/dashboard/revenue-chart.tsx:

```
import { generateYAxis } from '@/app/lib/utils';
…
import { fetchRevenue } from '@/app/lib/data';  //DM we'll call fetchRevenue below
…
//DM export default async function RevenueChart({
//DM  revenue,
//DM}: {
//DM  revenue: Revenue[];
//DM }) {

export default async function RevenueChart() {  //DM Make component async, remove the props
  const revenue = await fetchRevenue();         //DM Fetch data inside the component
…
```

Потоковый вывод компонента <RevenueChart> (5)

2	**Open and refresh localhost:3000/dashboard**	**Обновите страницу http://localhost:3000\dashboard Страница отображается почти мгновенно, в то время как для <RevenueChart> выводится заставка (можете замедлить выполнение fetchRevenue(), см. Slide 36: Приложение. Задерживаем запуск запроса в fetchRevenue())**



Потоковый вывод компонента <LatestInvoices>

1	Edit the page.tsx file	Отредактируйте файл page.tsx: 1. Удалите все экземпляры fetchLatestInvocies() и его данные из /dashboard/(overview)/page.tsx в компонент <LatestInvoices> 2. Импортируйте <Suspense> из React и оберните его вокруг <LatestInvoices />. 3. Передайте ему fallback-компонент <LatestInvociesSkeleton>.
1a	Copy the page.tsx file	(Или) скопируйте файл page.tsx: cd '(overview)' cp page.tsx.02.StreamingLatestInvoices page.tsx
2	Review the page.tsx file	Изучите файл page.tsx cat page.tsx

Потоковый вывод компонента <LatestInvoices> (2)

Убираем fetchLatestInvoices() в тело <LatestInvoices /> и оборачиваем его в <Suspense> с заставкой <LatestInvoicesSkeleton>. Вызываем <LatestInvoices> без параметров (props)

/app/dashboard/'(overview)'/page.tsx:

```
import { Card } from '@/app/ui/dashboard/cards';
…
import { fetchCardData } from '@/app/lib/data'; //DM fetchLatestInvoices removed
import { Suspense } from 'react';
import {
  RevenueChartSkeleton,
  LatestInvoicesSkeleton,
} from '@/app/ui/skeletons';   //DM <LatestInvoicesSkeleton> added

export default async function Page() {
  //DM const latestInvoices = await fetchLatestInvoices(); // moved to <LatestInvoices />
…
      <Suspense fallback={<LatestInvoicesSkeleton />}>
        <LatestInvoices />
      </Suspense>
    </div>
```

Потоковый вывод компонента <LatestInvoices> (3)

3	Edit the latest-invoices.tsx file	Отредактируйте файл app/ui/dashboard/latest-invoices.tsx: Обновите компонент <LatestInvoices> так, чтобы он извлекал свои данные с fetchLatestInvoices() и удалите его prop
3a	Copy the latest-invoices.tsx file	(Или) скопируйте файл latest-invoices.tsx: cd app/ui/dashboard cp latest-invoices.tsx.01.StreamingLatestInvoices latest-invoices.tsx
4	Review the latest-invoices.tsx file	Изучите файл latest-invoices.tsx cat latest-invoices.tsx

Потоковый вывод компонента <LatestInvoices> (4)

Корректируем заголовок LatestInvoices (убираем props), импортируем и вызываем fetchLatestInvoices() в теле функции LatestInvoices() {...}

/app/ui/dashboard/latest-invoices.tsx:

```
import { ArrowPathIcon } from '@heroicons/react/24/outline';
…
import { fetchLatestInvoices } from '@/app/lib/data'; //DM we'll call fetchLatestInvoices below

//DM export default async function LatestInvoices({
//DM  latestInvoices,
//DM}: {
//DM  latestInvoices: LatestInvoice[];
//DM}) {

export default async function LatestInvoices() { //DM Removed props above
  const latestInvoices = await fetchLatestInvoices(); //DM get latestInvoices here
…
```

Потоковый вывод компонента <LatestInvoices> (5)

5	**Open and refresh localhost:3000/dashboard**	**Обновите страницу http://localhost:3000\dashboard Страница отображается почти мгновенно, на месте <LatestInvoices> кратковременно выводится заставка** **NB. Для более длительного отображения заставки на экране можете замедлить функцию fetchLatestInvoices( ) таймером, аналогично тому, как это было сделано для fetchRevenue() (см. Slide 38: Приложение. Задерживаем запуск запроса в fetchRevenue())**

Эффективная группировка и вывод однородных компонентов

Проблема:

Теперь нам нужно обернуть компоненты <Card> в <Suspense>. Извлечение данных для каждой отдельной карты может привести к неприятному эффекту “выскакивания” при отрисовке.

Решение:

Вы можете сгруппировать карты с помощью компонента-обертки. Это приведет к тому, что статический <SideNav/> будет показан первым, а затем ***одновременно*** появятся все карты

Потоковый вывод компонента <CardWrapper>

1	Edit the page.tsx file	Отредактируйте файл page.tsx: 1. Удалите все компоненты <Card> 2. Удалите функцию fetchCardData() 3. Импортируйте <CardWrapper /> 4. Импортируйте компонент-заставку <CardsSkeleton /> 5. Оберните <CardWrapper / > в <Suspense>.
1a	Copy the page.tsx file	(Или) скопируйте файл page.tsx: cd '(overview)' cp page.tsx.03.StreamingCardWrapper page.tsx
2	Review the page.tsx file	Изучите файл page.tsx cat page.tsx

Потоковый вывод компонента <CardWrapper> (2)

Убираем fetchCardData() в тело <CardWrapper /> и оборачиваем его в <Suspense> с заставкой <CardsSkeleton>

/app/dashboard/’(overview)’/page.tsx:

```
import CardWrapper from '@/app/ui/dashboard/cards'; //DM <CardWrapper> is used instead of <Card>
import RevenueChart from '@/app/ui/dashboard/revenue-chart';
…
import {
  RevenueChartSkeleton,
  LatestInvoicesSkeleton,
  CardsSkeleton,
} from '@/app/ui/skeletons';   //DM <CardsSkeleton> added
…
//DM  const {
//DM    numberOfInvoices,
//DM    numberOfCustomers,
//DM    totalPaidInvoices,
//DM    totalPendingInvoices,
//DM  } = await fetchCardData(); //DM fetchCardData moved to <CardWrapper />
  return (
…
 <div className="grid gap-6 sm:grid-cols-2 lg:grid-cols-4">
       <Suspense fallback={<CardsSkeleton />}>
         <CardWrapper />
       </Suspense>
```

Потоковый вывод компонента <CardWrapper> (3)

3	Edit the cards.tsx file	Отредактируйте файл app/ui/dashboard/cards.tsx: 1. Импортируйте функцию fetchCardData() и вызовите ее внутри компонента <CardWrapper/> 2. Раскомментируйте весь необходимый код в этом компоненте
3a	Copy the cards.tsx file	(Или) скопируйте файл cards.tsx: cd app/ui/dashboard cp cards.tsx.01.StreamingCardWrapper cards.tsx
4	Review the cards.tsx file	Изучите файл cards.tsx cat cards.tsx

Потоковый вывод компонента <CardWrapper> (4)

Импортируем и вызываем fetchCardData() в теле функции CardWrapper() {...}. Раскомментируем код с <Card>s

/app/ui/dashboard/cards.tsx:

```
…
import { fetchCardData } from '@/app/lib/data'; //DM we'll use fetchCardData() below
…
export default async function CardWrapper() {
  const {
    numberOfInvoices,
    numberOfCustomers,
    totalPaidInvoices,
    totalPendingInvoices,
  } = await fetchCardData(); //DM call fetchAcrdData() right here
  return (
…
//DM Uncomment the code that contains <Card> elements below
```

Потоковый вывод компонента <CardWrapper> (5)

5	**Open and refresh localhost:3000/dashboard**	**Обновите страницу** http://localhost:3000\dashboard **Страница отображается почти мгновенно, и все карты загружаются одновременно.** **NB, Вы можете воспользоваться данным шаблоном для *одновременной* визуализации нескольких компонентов**
6	**Stop and remove the project environment**	**Остановите и удалите тестовую среду проекта** Slide 39: Приложение. Останов и очистка рабочей среды проекта

Оптимальное расположение границ <Suspense>

- Вы можете выводить страницу целиком, как мы это сделали с loading.tsx, но это может привести к более длительной отрисовке, если один из компонентов имеет существенно медленную загрузку данных
- Вы можете транслировать каждый компонент по отдельности, но это может вызвать неравномерную, прерывистую визуализацию пользовательского интерфейса на экране
- Вы также можете создать “ступенчатый” эффект, выводя страницу по разделам, но вам потребуется для этого создать компоненты-оболочки

В целом, хорошей практикой является перемещение функций выборки данных “вниз” по иерархии к компонентам, которым это нужно, а затем обертывание этих компонентов в <Suspense>. Тем не менее, нет ничего плохого в трансляции разделов или всей страницы, если именно это требуется вашему приложению.

Не бойтесь экспериментировать и выбирать то, что работает наилучшим образом: <Suspense> это мощной средство, которое поможет создать более приятный, доброжелательный пользовательский интерфейс.

Подведем итоги

1	**Изучили потоковый вывод и варианты его применения**
2	**Реализовали потоковый вывод средствами loading.tsx и <Suspense>**
3	**Поработали заставками - “скелетами загрузки” (loading skeletons)**
4	**Использовали группы маршрутов (route groups) для ограничения области действия loading.tsx**
5	**Узнали, где следует размещать границы <Suspense>**

Следующая лекция будет посвящена “partial prerendering”, новой модели визуализации Next.js, основанной на представленном здесь потоковом выводе

Следующая лекция -
10. Partial Pre-rendering /
Частичная предварительная отрисовка

Презентация доступна для скачивания здесь:

https://dmpsy.club/references/NextJS/lesson_009_streaming_rus.pdf

Загрузить приложение с GitHub:
https://github.com/DmitryMakar/nextjs-lesson-009.git

Поддержать автора: https://www.donationalerts.com/r/dmitrymak

Приложение А. Запуск рабочей среды проекта

Prerequisite	**Docker installed on your host**
To verify	**docker –v**
Example output	**Docker version 28.1.1, build 4eba377**
Postgres User	**nexxtapp-db**
Postgres Password	**NextJS**
Postgres DB	**nexxtapp-db**

1	**Launch the project testbed**	**Клонируйте проект и перейдите в папку scripts:** **git clone https://github.com/DmitryMakar/nextjs-lesson-009.git** **cd nextjs-lesson-009/scripts** **mkdir pgadmin && sudo chown -R 5050:80 pgadmin # set permissions** **docker compose up -d # start the environment** **docker compose logs # inspect logs** **docker container ls -a # check that pgAdmin and DB containers started ok**

Приложение А. Запуск рабочей среды проекта (2)

2	Configure pgAdmin 4	Откройте http://localhost:5050 # или по IP: http://<Your_(VM)_IP_address>:5050 Установите master password: <Your_password> Нажмите Add a new server и укажите: Name: test Host name/address: postgres_container Port: 5432 Maintenance DB: nexxtapp-db Username: nexxtapp-db Password: NextJS
3	Install node modules	Установите необходимые пакеты (node modules): cd nextjs-lesson-009 pnpm install Если потребуется, подтвердите установку доп. пакетов: pnpm approve-builds

Приложение А. Запуск рабочей среды проекта (3)

4	**Copy the PostgreSQL credentials to the .env file**	**Скопируйте учетные данные PostgreSQL в файл .env:**
		`cp .env.example .env`
		`cat .env  # verify it:`
		`POSTGRES_HOST=localhost`
		`POSTGRES_USER=nexxtapp-db`
		`POSTGRES_PASSWORD=NextJS`
		`POSTGRES_DATABASE=nexxtapp-db`
		`POSTGRES_PORT=5437`
		NB Соединяемся с БД “извне” контейнеров, поэтому localhost:5437, как это указано в docker-compose.yaml (5437:5432). Это сделано во избежание конфликта с локальной БД Postgres, если она уже установлена на хосте и слушает стандартный порт 5432

Приложение А. Запуск рабочей среды проекта (4)

5	**Seed the PostgreSQL with initial data**	**Заполните схему БД исходными данными:** pnpm seed > @ seed /home/dmitry-makarenkov/NextJSTraining/nextjs-lesson-009 > node -r dotenv/config ./scripts/seed.mjs Created "users" table Seeded 1 users Created "customers" table Seeded 6 customers Created "invoices" table Seeded 13 invoices Created "revenue" table Seeded 12 revenue
6	**Check the data seeded via pgAdmin**	**Проверьте, что данные поступили в БД,** **Откройте http://localhost:5050/browser/ # или по IP:** **http://<Your_(VM)_IP_address>:5050/browser/** Tools | Query tool select * from book; select * from users; select * from customers; select * from invoices; select * from revenue;

Приложение А. Запуск рабочей среды проекта (5)

7	**Start the dev server**	**Запустите сервер разработки:** **pnpm run dev** **> next dev** **▲ Next.js 14.2.5** **- Local: http://localhost:3000** **- Environments: .env** **✓ Starting...** **✓ Ready in 14.8s**
8	**Verify it works**	**Проверьте, что приложение открывается;** **Откройте http://localhost:3000/dashboard # или по IP:** **http://<Your_(VM)_IP_address>:3000/dashboard/** **В логах отобразится, примерно, следующее:** **○ Compiling /dashboard ...** **✓ Compiled /dashboard in 1970ms (618 modules)** **GET /dashboard 200 in 6420ms**

Вернуться к Slide 6: Потоковый вывод страницы dashboard.tsx

Приложение B. Останов и очистка рабочей среды проекта

1	**Clean the project when done**	**Закройте web-страницы приложения и pgAdmin’a** **Остановите сервер разработки (Ctrl-C) и очистите среду:** **cd scripts** **docker compose down** **sudo rm -rf pgadmin pgdata** **docker image rm dpage/pgadmin4:latest postgres:15-alpine**

Вернуться к Slide 30: Потоковый вывод компонента <CardWrapper> (5)

Приложение C. Задерживаем запуск запроса в ***fetchRevenue()***

0	**Prepare fetchRevenue() for app/lib/data.ts**	**fetchRevenue() отредактирована в файле app/lib/data.ts.slowedFetchRevenue:**
		…
		await client.connect();
		
		//Delay a response for demo purposes
		console.log("Fetching revenue data...");
		await new Promise((resolve) => setTimeout(resolve, 10000));
		
		const result = await client.query('select * from revenue');
		console.log(result.rows);
		console.log('Data fetch completed after 10 seconds.');
		await client.end();
		
		return result.rows;
		…

Вернуться к Slide 19: Потоковый вывод компонента <RevenueChart> (5)

Slide 24: Потоковый вывод компонента <LatestInvoices> (5)

https://dmpsy.club
postmaster@dmspy.club

https://www.donationalerts.com/r/dmitrymak