Курс NextJS. Статический или динамический рендеринг?

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

1	The “slow query” problem. Statement of the problem	Проблема “медленного запроса”. Постановка задачи
2	Delay fetchRevenue()	Задерживаем выполнение fetchRevenue()
3	Preliminary steps	Подготовительные шаги
4	Dev-server demo results	Результаты демонстрации на dev-сервере
5	Production demo results	Результаты демонстрации в production
6	Static rendering pros. & cons.	Особенности статической отрисовки
7	Dynamic rendering pros. & cons.	Особенности динамической отрисовки
8	Summary	Подведем итоги

Официальная версия от NextJS: https://nextjs.org/learn/dashboard-app/static-and-dynamic-rendering

Проблема “медленного запроса” из прошлой лекции. Постановка задачи

В прошлой лекции мы научились преодолевать нежелательное последовательное или “каскадное” (waterfall) выполнение запросов за счет parallel data fetching, когда эти запросы к БД запускаются одновременно (используем ***Promise*** для запуска), но не ответили на вопрос:

Что произойдет с отрисовкой страницы, если один из запросов выполняется особенно долго? Можно предположить, что скорость отрисовки страницы будет определяться именно скоростью выполнения самого медленного запроса. Для проверки этой версии искусственно задержим выполнение запроса в функции ***fetchRevenue()***

Вторая задача - изучить особенности и возможные ограничения сборки (compiling), хранения и загрузки в Web Browser статического контента в NextJS. Всегда ли хорош pre-rendering? Какова альтернатива статической отрисовке (static rendering)?

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;
		…

Подготовительные шаги

Prerequisite	PostgreSQL installed on your host
To verify	pg_config –version
Example output	PostgreSQL 16.8 (Ubuntu 16.8-0ubuntu0.24.04.1)
Postgres User	nexxtapp-db
Postgres Password	<nexxtapp-db_password>
Postgres DB	nexxtapp-db

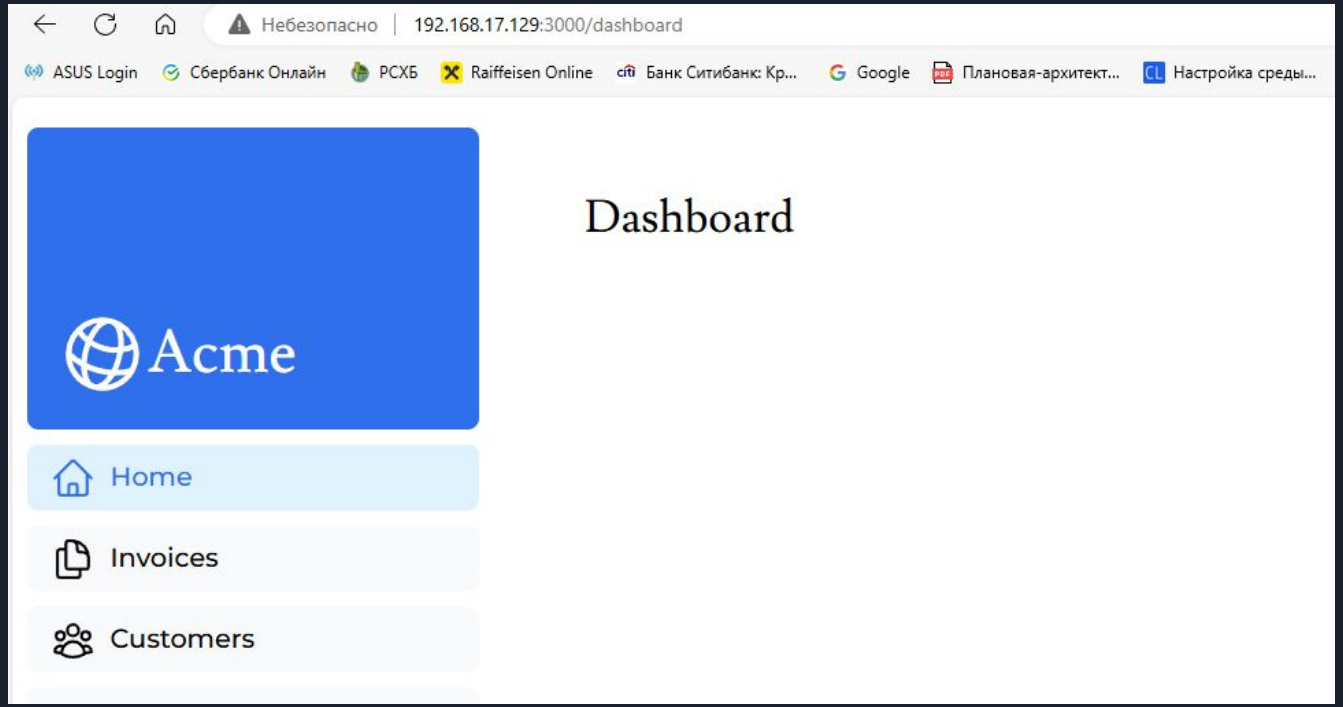
1	Clone the application	Клонируйте проект и перейдите в его папку: git clone https://github.com/DmitryMakar/nextjs-lesson-008.git cd nextjs-lesson-008
2	Install node modules	Установите необходимые пакеты (node modules): pnpm install Если потребуется, подтвердите установку доп. пакетов: pnpm approve-builds

Подготовительные шаги (2)

3	Add DB credentials to .env	Укажите данные для соединения с БД PostgreSQL в файле .env в корневой папке проекта: nano .env POSTGRES_HOST=localhost POSTGRES_USER=nexxtapp-db POSTGRES_PASSWORD=<nexxtapp-db_password> POSTGRES_DATABASE=nexxtapp-db Ctrl-X # сохранить изменения и закрыть nano
4	Seed the DB with data	Запустите скрипт загрузки данных в БД: pnpm seed
5	Delay fetchRevenue() execution by 10 seconds	Скопируйте файл с “замедленным запросом” в data.ts: cd app/lib # из корневой папки проекта cp data.ts.slowedFetchRevenue data.ts

Подготовительные шаги (3)

6	**Launch nexxt-app and open localhost:3000/dashboard**	**Запустите приложение на dev-сервере:** **pnpm dev # из корневой папки проекта** **Откройте http://localhost:3000\dashboard**



Результаты демонстрации на сервере разработки (dev)

1. Страницы компилируются (“Compiling”) только при первом обращении к ним, хранятся на dev-сервере без изменений и предоставляются по запросу GET от веб-браузера
2. Предоставленные в браузер страницы хранятся им и выдаются из cache без дополнительного обращения (GET) на dev-сервер, что существенно ускоряет отображение информации
3. Веб браузер инициирует запрос GET в сторону dev-сервера только в случае принудительного обновления (Refresh) существующей страницы или прямого запроса к этой странице через командую строку браузера
4. При получении запроса GET, dev-сервер, как правило, не выполняет запрос к БД, а выдает уже имеющуюся на хранении, изначально скомпилированную страницу

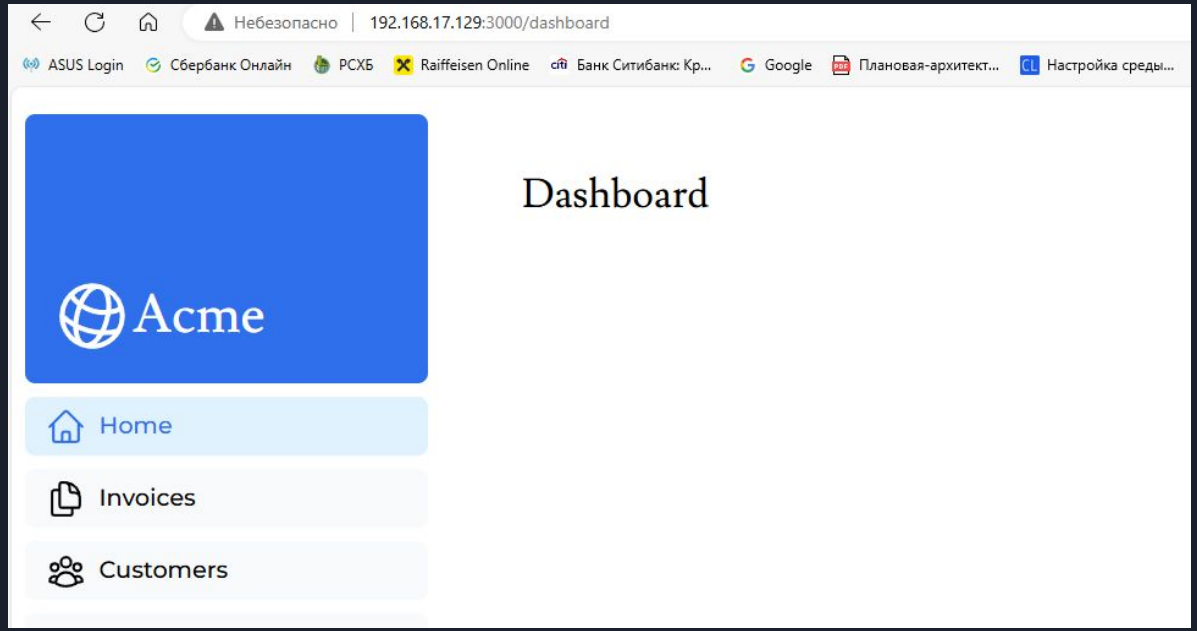
Таким образом, обеспечивается максимально высокая скорость работы со страницами, но данные из БД, если и актуализируются, то недостаточно часто (раз в несколько минут), да и то исключительно при принудительном обновлении страницы

Подготовительные шаги (4)

7	Stop dev server and remove the ./next folder	Остановите dev-server: Ctrl-C Удалите папку сборок (опционально): rm -rf .next
8	Create a production build	Создайте эксплуатационную (production) сборку: pnpm run build Обратите внимание на лог сборки: 1. Все страницы созданы как статические 2. Запрос из fetchRevenue() выполняется однократно, при сборке страницы, и его результаты размещаются в ней на все время ее существования, какие-либо обновления данных при этом принципиально невозможны

Подготовительные шаги (5)

9	**Launch the production build and open localhost:3000/dashboard**	**Запустите сборку, убедитесь в работоспособности и высокой скорости отображения статических страниц** **pnpm start** **Откройте http://localhost:3000\dashboard** **Ctrl-C # остановить production-сервер**



Результаты демонстрации эксплуатационного (optimized prod) варианта

1. Страницы компилируются (“Compiling”) однократно, при выполнении скрипта ***pnpm run build***, хранятся на prod-сервере без изменений и предоставляются по запросу GET от веб-браузера
2. Предоставленные в браузер страницы хранятся им и выдаются из cache без дополнительного обращения (GET) на prod-сервер, что существенно ускоряет отображение информации
3. Веб браузер инициирует запрос GET в сторону prod-сервера только в случае принудительного обновления (Refresh) существующей страницы или прямого запроса к этой странице через командную строку браузера
4. При получении запроса GET, prod-сервер ***не выполняет запрос к БД***, а выдает уже имеющуюся на хранении, изначально скомпилированную страницу

Таким образом, обеспечивается максимально высокая скорость работы со страницами, но ***данные из БД, отображаемые на страницах, не актуализируются с момента сборки этих страниц***

Особенности статической отрисовки (static rendering)

Выборка и отрисовка данных происходят на сервере во время сборки или при повторной валидации данных. Всякий раз пользователю предоставляется неизменный кэшированный результат. Преимущества static rendering:

- ***Более высокая производительность веб-сайтов*** за счет кэширования в браузере и хранения готовых, неизменных страниц на веб-сервере
- ***Уменьшенная нагрузка на веб-сервер*** — поскольку контент кэшируется, веб-серверу нет необходимости динамически генерировать контент для каждого запроса пользователя, что снижает затраты на вычисления.
- ***Преимущества SEO-продвижения*** — статический контент проще индексировать поисковым роботам, поскольку контент уже доступен при загрузке страницы, что может повысить рейтинг сайта в поисковых системах.

Static rendering хорош для пользовательского интерфейса без изменяемых данных например, для сообщений в блоге или описания продуктов в интернет-магазине.

Static rendering не подойдет для панелей мониторинга с данными и графиками, обновляемыми в реальном времени. В подобных случаях следует использовать альтернативу static rendering - динамическую отрисовку или визуализацию - dynamic rendering.

Особенности динамической отрисовки (dynamic rendering)

Контент актуализируется на веб-сервере при каждом посещении страницы. Преимущества dynamic rendering:

- ***Вывод данных в реальном времени*** - динамический рендеринг позволяет вашему приложению отображать данные в реальном времени или часто обновляемые данные. Это идеально подходит для приложений, где данные постоянно изменяются
- ***Работа с персонализированным контентом*** - удобно обслуживать например, профили пользователей и обновлять данные в процессе общения с ними
- ***Использование информации из запроса*** - динамический рендеринг позволяет вам получать доступ к информации, которая становится известной только во время запроса, например, к файлам cookie или параметрам поиска URL

О реализации ***dynamic rendering*** в NextJS см. следующие лекции

Подведем итоги

1	**Убедились в том, что задержка (первоначальной) отрисовки страницы определяется скоростью выполнения ее наиболее “медленного” запроса (функции), что может вызвать негативный user experience**
2	**Изучили преимущества (быстрый и надежный вывод кэшированных страниц) и недостатки (невозможность обновления данных в реальном времени) статической отрисовки**
3	**Поняли, что для быстро меняющихся данных необходимо использовать динамическую отрисовку (dynamic rendering), когда страница обновляется при каждом визите пользователя**

Полезная информация

Сборки, порождаемые командами

pnpm run dev
pnpm run build

располагаются в скрытой папке *.next*
в корневом каталоге проекта

Следующие лекции -
9. Streaming in NextJS / Потоки данных в NextJS
8d. Containerization / Эскиз контейнеризации

Презентация доступна для скачивания здесь:

https://dmpsy.club/references/NextJS/lesson_008_static_and_dynamic_rendering_rus.pdf

Загрузить приложение с GitHub:
https://github.com/DmitryMakar/nextjs-lesson-008

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

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

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