Лабораторная работа 3: Разработка SPA (Single Page Application)

Студент: Даньшин Семён
Группа: K3440
Дата: 12 января 2026 г.

Тема работы

Разработка одностраничного веб-приложения (SPA) с использованием фреймворка. **Примечание:** В соответствии с архитектурой проекта, вместо Vue.js используется React (Next.js), что является аналогичным современным решением для построения SPA.

Вариант задания

Вариант 6: Свой вариант (Мультибанкинговая платформа)

Реализация SPA архитектуры

Проект реализован как SPA с использованием **Next.js App Router**. Это обеспечивает мгновенные переходы между страницами без полной перезагрузки браузера, сохраняя при этом состояние приложения.

1. Маршрутизация (Routing)

В Next.js используется файловая маршрутизация в папке `app/`.

- `app/page.tsx` -> Главная страница (`/`)
- `app/auth/page.tsx` -> Страница авторизации (`/auth`)
- `app/history/page.tsx` -> История операций (`/history`)
- `app/payments/page.tsx` -> Платежи (`/payments`)

В отличие от Vue Router, где маршруты описываются в js-файле конфигурации, здесь структура папок определяет URL. Навигация осуществляется через компонент `<Link>` или хук `useRouter`.

```
import { useRouter } from "next/navigation";

// Программная навигация
const router = useRouter();
router.push("/");
```

2. Компонентный подход

Интерфейс разбит на независимые компоненты, аналогично Vue Components.

- **Component Tree:**

    - `RootLayout`
        - `Providers` (Context)
            - `AuthGuard`
            - `Navbar`
            - `AssistantSidebar`
            - `Page Content` (динамический)
            - `BottomNavigation`

Используются функциональные компоненты React с хуками (`useState`, `useEffect`), что соответствует Composition API во Vue 3.

```
// Пример компонента (аналог Vue Setup Script)
export const BottomNavigation = () => {
  const pathname = usePathname();
  const { openFromBottom } = useDrawer(); // Custom Hook

  return (
    // JSX (Template)
    <div className="...">...</div>
  );
};
```

3. Управление состоянием (State Management)

Вместо Vuex/Pinia используется **React Context API**, который позволяет передавать данные через дерево компонентов без "prop drilling".

Реализованные контексты:

1. **ChatContext:** Управляет состоянием чата с ИИ-ассистентом (сообщения, статус загрузки, активная сессия).
    - *Глобальное состояние:* `messages`, `isLoading`, `chatId`.
    - *Логика:* Отправка сообщений, получение ответов, интеграция с API.
2. **DrawerContext:** Управляет видимостью боковых панелей (ассистент, меню).
    - Позволяет открыть панель ассистента из любой кнопки в приложении.
3. **SyncContext:** Управляет статусом синхронизации данных.
    - Позволяет триггерить обновление данных на дашборде из других компонентов (например, после успешной оплаты в чате).

Пример Context Provider (`context/DrawerContext.tsx`):

```
export const DrawerProvider = ({ children }: { children: ReactNode }) => {
  const { isOpen, onOpen, onOpenChange } = useDisclosure();
  // ... логика состояния ...
```

```
  return (
    <DrawerContext.Provider value={{ isOpen, onOpen, ... }}>
      {children}
      {/* Глобальный компонент Drawer, доступный для управления отовсюду
*/}
      <AssistantDrawer isOpen={isOpen} ... />
    </DrawerContext.Provider>
  );
};
```

4. Взаимодействие компонентов (Custom Hooks / Composables)

Повторяющаяся логика вынесена в кастомные хуки (аналог Vue Composables).

- `useDrawer()`: Предоставляет методы `openFromBottom`, `openFromRight`.
- `useChat()`: Инкапсулирует сложную логику websocket/polling соединения с чатом.
- `useSyncContext()`: Доступ к токенам обновления.

5. Миграция с классической верстки

Приложение было изначально спроектировано как SPA, но включает элементы, улучшенные по сравнению с классической версткой (Lab 1):

- **Layout Persistence:** Навбар и сайдбар не перерисовываются при переходе между страницами.
- **Client-side Prefetching:** Страницы подгружаются заранее при наведении курсора на ссылку.
- **Shared State:** Данные авторизации и состояние чата сохраняются при навигации.

Заключение

В рамках третьей лабораторной работы было разработано полноценное SPA-приложение. Использование Next.js и React позволило создать быструю и отзывчивую систему с разделением на компоненты, глобальным управлением состоянием и клиентской маршрутизацией, что полностью удовлетворяет требованиям к современным frontend-приложениям.