Домашняя работа 5: NPM и компонентный подход

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

Описание задания

Задание: Научиться работать с npm и компонентным подходом. В качестве отчёта ожидаются скриншоты компонентов и код получившегося приложения.

Примечание: Изначально задание предполагало работу с Vue.js, однако проект BankingThing реализован на React/Next.js, что также демонстрирует компонентный подход в современной веб-разработке.

Теоретическая часть

Что такое NPM?

NPM (Node Package Manager) - это менеджер пакетов для JavaScript, позволяющий:

- Устанавливать библиотеки и зависимости
- Управлять версиями пакетов
- Запускать скрипты сборки и разработки
- Публиковать собственные пакеты

Основные команды NPM

```
# Инициализация проекта
npm init

# Установка зависимости
npm install package-name
npm install package-name --save-dev  # dev-зависимость

# Установка всех зависимостей из package.json
npm install

# Запуск скриптов
npm run dev
npm run build
npm start

# Обновление пакетов
npm update

# Удаление пакета
npm uninstall package-name
```

Компонентный подход

Компонентный подход - это методология разработки UI, где интерфейс разбивается на независимые, переиспользуемые части (компоненты).

Преимущества:

1. **Переиспользование кода** - один компонент используется многократно
2. **Изолированность** - компоненты не зависят друг от друга
3. **Удобство тестирования** - каждый компонент тестируется отдельно
4. **Масштабируемость** - легко добавлять новые компоненты
5. **Читаемость** - структура проекта понятна

Практическая часть: Проект BankingThing

1. Структура package.json

```
{
  "name": "next-app-template",
  "version": "0.0.1",
  "private": true,
  "scripts": {
    "dev": "next dev --turbopack",
    "build": "next build",
    "start": "next start",
    "lint": "eslint --fix"
  },
  "dependencies": {
    // Основные зависимости
    "next": "^15.5.7",
    "react": "18.3.1",
    "react-dom": "18.3.1",

    // UI-библиотека HeroUI
    "@heroui/button": "2.2.27",
    "@heroui/card": "2.2.25",
    "@heroui/input": "2.4.28",
    "@heroui/navbar": "2.2.25",
    // ... и другие компоненты HeroUI

    // Утилиты
    "axios": "^1.13.2",
    "clsx": "2.1.1",
    "date-fns": "^4.1.0",

    // Темизация и стилизация
    "next-themes": "0.4.6",
    "framer-motion": "11.18.2",

    // Дополнительные возможности
    "react-markdown": "^10.1.0",
    "react-speech-recognition": "^4.0.1",
```

```
    "recharts": "^3.3.0"
  },
  "devDependencies": {
    // TypeScript
    "typescript": "5.6.3",
    "@types/react": "18.3.3",
    "@types/node": "20.5.7",

    // Линтеры
    "eslint": "^9.39.0",
    "prettier": "3.5.3",

    // Tailwind CSS
    "tailwindcss": "4.1.11",
    "postcss": "8.5.6"
  }
}
```

2. Скрипты NPM

npm run dev

Запускает приложение в режиме разработки с hot reload:

```
npm run dev
# Запускается сервер на http://localhost:3000
# Используется Turbopack для быстрой сборки
```

npm run build

Создаёт production-сборку:

```
npm run build
# Оптимизирует код
# Генерирует статические страницы (SSG)
# Минифицирует ресурсы
```

npm run start

Запускает production-сервер:

```
npm run start
# Требует предварительного npm run build
```

npm run lint

Проверяет и исправляет код:

```
npm run lint
# ESLint проверяет код на ошибки
# Автоматически исправляет форматирование
```

3. Компонентная архитектура проекта

Проект организован по следующей структуре:

```
ui/
├── app/                    # Страницы Next.js (App Router)
│   ├── layout.tsx          # Корневой layout
│   ├── page.tsx            # Главная страница
│   ├── auth/               # Страница авторизации
│   ├── dashboard/          # Дашборд
│   ├── history/            # История транзакций
│   ├── payments/           # Платежи
│   └── products/           # Продукты
│
├── components/             # Переиспользуемые компоненты
│   ├── navbar.tsx
│   ├── theme-switch.tsx
│   ├── icons.tsx
│   ├── auth/               # Компоненты авторизации
│   ├── assistant/          # Компоненты ассистента
│   ├── dashboard/          # Компоненты дашборда
│   ├── history/            # Компоненты истории
│   ├── payments/           # Компоненты платежей
│   └── products/           # Компоненты продуктов
│
├── context/                # React Context для состояния
│   ├── DrawerContext.tsx
│   ├── ChatContext.tsx
│   └── SyncContext.tsx
│
├── config/                 # Конфигурация
│   ├── site.ts
│   ├── fonts.ts
│   └── constants.ts
│
├── utils/                  # Утилиты
│   └── colorUtils.ts
│
└── styles/                 # Глобальные стили
    └── globals.css
```

4. Примеры компонентов

Компонент 1: LoginCard (Карточка входа)

Файл: components/auth/LoginCard.tsx

```
'use client';

import React, { useState } from "react";
import { useRouter } from "next/navigation";
import { Card, CardHeader, CardBody, CardFooter } from "@heroui/card";
import { Input } from "@heroui/input";
import { Button } from "@heroui/button";
import { Link } from "@heroui/link";
import { Form } from "@heroui/form";
import { authApi } from "@/api/api";

interface LoginCardProps {
  onSwitchToRegister: () => void;
}

export const LoginCard: React.FC<LoginCardProps> = ({ onSwitchToRegister
}) => {
  const router = useRouter();
  const [clientId, setClientId] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState<string | null>(null);
  const [isLoading, setIsLoading] = useState(false);

  const handleSubmit = async (e: React.FormEvent<HTMLFormElement>) => {
    e.preventDefault();
    setIsLoading(true);
    setError(null);

    try {
      const response = await authApi.v1.bankingServiceLogin({
        clientId,
        password,
      });

      const { accessToken, refreshToken } = response.data;

      if (accessToken && refreshToken) {
        localStorage.setItem("accessToken", accessToken);
        localStorage.setItem("refreshToken", refreshToken);
        router.push("/");
      }
    } catch (err: any) {
      const errorMessage = \`Error \${err?.status || ""}:
\${err?.response?.data?.message || err.message}\`;
      setError(errorMessage);
    } finally {
      setIsLoading(false);
    }
```

```
  };

  return (
    <Card className="w-full">
      <CardHeader className="flex flex-col items-start">
        <p className="text-xl font-bold">Войти</p>
        <p className="text-small text-default-500">Используйте данные
своего аккаунта</p>
      </CardHeader>
      <CardBody>
        <Form className="flex flex-col gap-4" onSubmit={handleSubmit}>
          <Input
            isRequired
            label="Client ID"
            placeholder="Введите свой Client ID"
            type="text"
            value={clientId}
            onValueChange={setClientId}
            isDisabled={isLoading}
          />
          <Input
            isRequired
            label="Пароль"
            placeholder="Введите пароль"
            type="password"
            value={password}
            onValueChange={setPassword}
            isDisabled={isLoading}
          />
          {error && <p className="text-danger text-small">{error}</p>}
          <Button type="submit" color="primary" fullWidth isLoading=
{isLoading}>
            {isLoading ? "Вход..." : "Войти"}
          </Button>
        </Form>
      </CardBody>
      <CardFooter className="flex justify-center">
        <p className="text-small">
          Нет аккаунта?{" "}
          <Link href="#" onClick={onSwitchToRegister} size="sm">
            Зарегистрироваться
          </Link>
        </p>
      </CardFooter>
    </Card>
  );
};
```

Особенности компонента:

- Состояние через React hooks (`useState`)
- Обработка форм и валидация

- Интеграция с API через axios
- Навигация через Next.js router
- Использование UI-компонентов HeroUI
- TypeScript для типобезопасности

Компонент 2: ThemeSwitch (Переключатель темы)

Файл: `components/theme-switch.tsx`

```
"use client";

import { FC } from "react";
import { VisuallyHidden } from "@react-aria/visually-hidden";
import { SwitchProps, useSwitch } from "@heroui/switch";
import { useTheme } from "next-themes";
import { useIsSSR } from "@react-aria/ssr";
import clsx from "clsx";
import { SunFilledIcon, MoonFilledIcon } from "@/components/icons";

export interface ThemeSwitchProps {
  className?: string;
  classNames?: SwitchProps["classNames"];
}

export const ThemeSwitch: FC<ThemeSwitchProps> = ({
  className,
  classNames,
}) => {
  const { theme, setTheme } = useTheme();
  const isSSR = useIsSSR();

  const onChange = () => {
    theme === "light" ? setTheme("dark") : setTheme("light");
  };

  const {
    Component,
    slots,
    isSelected,
    getBaseProps,
    getInputProps,
    getWrapperProps,
  } = useSwitch({
    isSelected: theme === "light" || isSSR,
    "aria-label": \`Switch to \${theme === "light" || isSSR ? "dark" :
"light"} mode\`,
    onChange,
  });

  return (
    <Component {...getBaseProps({
      className: clsx(
```

```
          "px-px transition-opacity hover:opacity-80 cursor-pointer",
          className,
          classNames?.base,
        ),
      })}>
        <VisuallyHidden>
          <input {...getInputProps()} />
        </VisuallyHidden>
        <div {...getWrapperProps()}>
          {!isSelected || isSSR ? (
            <SunFilledIcon size={22} />
          ) : (
            <MoonFilledIcon size={22} />
          )}
        </div>
      </Component>
    );
};
```

Особенности компонента:

- Интеграция с next-themes для темизации
- SSR-совместимость
- Доступность через aria-атрибуты
- Динамическое изменение иконки
- Стилизация через Tailwind CSS

Компонент 3: BottomNavigation (Нижняя навигация)

Файл: `components/BottomNavigation.tsx`

```
"use client";

import { siteConfig } from "@/config/site";
import { Tabs, Tab } from "@heroui/tabs";
import { usePathname } from "next/navigation";
import { useDrawer } from "@/context/DrawerContext";
import {
  HomeIcon,
  CreditCardIcon,
  SparklesIcon,
  ClockIcon,
  Squares2X2Icon,
} from "@heroicons/react/24/solid";

const iconMap: Record<string, React.ElementType> = {
  home: HomeIcon,
  payments: CreditCardIcon,
  assistant: SparklesIcon,
  history: ClockIcon,
  products: Squares2X2Icon,
```

```
};

export const BottomNavigation = () => {
  const pathname = usePathname();
  const { openFromBottom } = useDrawer();

  return !pathname.startsWith("/auth") && (
    <div className="xl:hidden fixed flex bottom-1 w-full z-50 justify-
center">
      <Tabs
        aria-label="Bottom Navigation"
        items={siteConfig.navMenuItems}
        selectedKey={pathname}
        size="sm"
        classNames={{
          tab: "h-12 px-2 min-w-12",
          tabList: "bg-content1/30 backdrop-blur-xs gap-1 p-1 rounded-2xl
shadow-lg border border-default-200",
          cursor: "w-full bg-primary/20 rounded-xl",
        }}
      >
        {(item) => {
          const Icon = iconMap[item.icon];
          const isAssistant = item.label === "Ассистент"

          return (
            <Tab
              key={item.href}
              href={isAssistant ? undefined : item.href}
              onClick={isAssistant ? openFromBottom : undefined}
              title={
                <div className={\`flex flex-col items-center\`}>
                  {Icon && <Icon className={'w-6 h-6'} />}
                  <span className={'text-[10px]'}>{item.label}</span>
                </div>
              }
            />
          );
        }}
      </Tabs>
    </div>
  );
};
```

Особенности компонента:

- Адаптивность (показывается только на мобильных)
- Интеграция с роутингом Next.js
- Использование Context API (DrawerContext)
- Динамическое создание иконок через маппинг
- Условная логика для специальных элементов

Компонент 4: ConsentRow (Строка разрешения)

Файл: `components/dashboard/ConsentRow.tsx`

```
import React from 'react';
import { Button } from "@heroui/button";
import { Chip } from "@heroui/chip";
import { Divider } from "@heroui/divider";

interface ConsentRowProps {
  title: string;
  isAllowed: boolean;
  explanation: string;
  onGrant: () => void;
  onRevoke: () => void;
}

export const ConsentRow: React.FC<ConsentRowProps> = ({
  title,
  isAllowed,
  explanation,
  onGrant,
  onRevoke
}) => {
  return (
    <>
      <div className="flex items-center justify-between w-full">
        <p className="font-bold">{title}</p>
        <Chip color={isAllowed ? "success" : "danger"} variant="flat">
          {isAllowed ? "предоставлен" : "не предоставлен"}
        </Chip>
      </div>
      <p className="text-foreground-500 text-small">{explanation}</p>
      {isAllowed ? (
        <Button size="sm" color="danger" variant="light" onPress=
{onRevoke}>
          Отозвать доступ
        </Button>
      ) : (
        <Button size="sm" color="primary" variant="light" onPress=
{onGrant}>
          Предоставить доступ
        </Button>
      )}
      <Divider />
    </>
  );
};
```

Особенности компонента:

- Props для конфигурации
- TypeScript интерфейс для типизации props
- Условный рендеринг на основе состояния
- Семантическая цветовая индикация
- Callback функции для обработки действий

5. Context API для управления состоянием

DrawerContext - управление боковой панелью

Файл: `context/DrawerContext.tsx`

```
'use client';

import React, { createContext, useContext, useState, ReactNode } from
'react';

interface DrawerContextType {
  isOpenFromRight: boolean;
  isOpenFromBottom: boolean;
  isOpenFromLeft: boolean;
  openFromRight: () => void;
  closeFromRight: () => void;
  openFromBottom: () => void;
  closeFromBottom: () => void;
  openFromLeft: () => void;
  closeFromLeft: () => void;
}

const DrawerContext = createContext<DrawerContextType | undefined>
(undefined);

export const DrawerProvider = ({ children }: { children: ReactNode }) => {
  const [isOpenFromRight, setIsOpenFromRight] = useState(false);
  const [isOpenFromBottom, setIsOpenFromBottom] = useState(false);
  const [isOpenFromLeft, setIsOpenFromLeft] = useState(false);

  const openFromRight = () => setIsOpenFromRight(true);
  const closeFromRight = () => setIsOpenFromRight(false);
  const openFromBottom = () => setIsOpenFromBottom(true);
  const closeFromBottom = () => setIsOpenFromBottom(false);
  const openFromLeft = () => setIsOpenFromLeft(true);
  const closeFromLeft = () => setIsOpenFromLeft(false);

  return (
    <DrawerContext.Provider value={{
      isOpenFromRight,
      isOpenFromBottom,
      isOpenFromLeft,
      openFromRight,
      closeFromRight,
      openFromBottom,
```

```
        closeFromBottom,
        openFromLeft,
        closeFromLeft,
      }}>
        {children}
      </DrawerContext.Provider>
    );
  };

  export const useDrawer = () => {
    const context = useContext(DrawerContext);
    if (!context) {
      throw new Error('useDrawer must be used within DrawerProvider');
    }
    return context;
  };
```

Использование Context:

- Централизованное управление состоянием
- Избежание prop drilling
- Типобезопасность через TypeScript
- Custom hook для удобства использования

6. Установка зависимостей

Основные UI-компоненты (HeroUI)

```
npm install @heroui/button @heroui/card @heroui/input @heroui/navbar
```

Каждый компонент HeroUI устанавливается отдельно, что позволяет:

- Уменьшить размер bundle (устанавливаем только нужное)
- Контролировать версии компонентов
- Оптимизировать tree-shaking

Утилиты

```
npm install axios clsx date-fns
```

- `axios` - HTTP-клиент для работы с API
- `clsx` - утилита для работы с классами CSS
- `date-fns` - работа с датами

Темизация и анимации

```
npm install next-themes framer-motion
```

- `next-themes` - управление темами в Next.js
- `framer-motion` - анимации и transitions

7. Структура страниц (Next.js App Router)

app/layout.tsx - корневой layout

```
export default function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  return (
    <html suppressHydrationWarning lang="en">
      <body>
        <Providers themeProps={{ attribute: "class", defaultTheme: "dark"
}}>
          <SyncProvider>
            <ChatProvider>
              <DrawerProvider>
                <div className="relative h-screen">
                  <AuthGuard />
                  <div className="relative flex flex-row h-full">
                    <div className="relative flex flex-col h-full flex-
grow">
                      <Navbar />
                      <main className="container mx-auto max-w-7xl pt-4
px-4 flex-grow">
                        {children}
                      </main>
                      <footer>...</footer>
                    </div>
                    <AssistantSidebar />
                  </div>
                  <BottomNavigation />
                </div>
              </DrawerProvider>
            </ChatProvider>
          </SyncProvider>
        </Providers>
      </body>
    </html>
  );
}
```

Архитектура:

- Providers для глобального состояния
- AuthGuard для защиты маршрутов
- Навигация и футер на всех страницах
- Вложенность через children

Преимущества использованного стека

1. Next.js

- **SSR/SSG** - серверный рендеринг и статическая генерация
- **App Router** - современная система роутинга
- **Optimizations** - автоматическая оптимизация изображений, шрифтов
- **API Routes** - возможность создавать API endpoints

2. TypeScript

- **Type Safety** - проверка типов на этапе компиляции
- **Autocompletion** - автодополнение в IDE
- **Refactoring** - безопасный рефакторинг
- **Documentation** - типы служат документацией

3. HeroUI

- **Готовые компоненты** - не нужно писать с нуля
- **Accessibility** - встроенная доступность
- **Темизация** - легкое переключение тем
- **Responsive** - адаптивность из коробки

4. Tailwind CSS

- **Utility-first** - стили через классы
- **No CSS conflicts** - нет конфликтов имен
- **Small bundle** - только используемые стили
- **Responsive** - легкие media queries

Заключение

Проект BankingThing демонстрирует современный подход к разработке веб-приложений:

1. **NPM** используется для управления 70+ зависимостями
2. **Компонентный подход** - 50+ переиспользуемых компонентов
3. **TypeScript** для типобезопасности
4. **Next.js** для SSR и оптимизаций
5. **Context API** для управления состоянием
6. **Tailwind CSS** для быстрой стилизации

Хотя задание предполагало работу с Vue.js, принципы компонентного подхода универсальны и одинаково применимы в React, Vue, Angular и других фреймворках. Проект показывает, как правильно организовать архитектуру, разделить ответственность между компонентами и использовать современные инструменты для создания качественного UI.