

مقدمه

ایریس نه تنها یک قالب وب معمولی است بلکه یک شاهکار دقیقاً طراحی شده است که در بازار پر از گزینه‌های طراحی ضعیف و عمومی برجسته می‌شود. هر جنبه از ایریس، از رابط کاربری ظریف آن تا معماری کد قوی، به دقت طراحی شده است تا انعطاف‌پذیری و مقیاس‌پذیری بی‌نظیر برای پروژه‌های شما تضمین کند.

برخلاف سایر قالب‌ها که به کتابخانه‌های منبع باز معمولی تکیه می‌کنند، ایریس مجموعه‌ای کامل از کامپوننت‌های رابط کاربری ساخته شده توسط خود را ارائه می‌دهد. این کامپوننت‌ها نه تنها از نظر عملکرد غنی هستند بلکه کنترل بیشتری نیز ارائه می‌دهند، به شما اجازه می‌دهد که هر جزئی را مطابق با نیازهای خاص خود تنظیم کنید.

ایریس با ویژگی‌های پیشرفته‌ای مانند پشتیبانی از چند زبان، حالت تاریک و روشن، چیدمان از راست به چپ، سفارشی کردن رنگ تم و توانایی تغییر بین شش طرح پیش‌فرض به راحتی مجهز شده است. چه شما یک پلتفرم جهانی می‌سازید یا یک برنامه نیش، انعطاف‌پذیری ایریس شما را پوشش می‌دهد.

علاوه بر این، نمونه‌های برنامه شامل شده که بر اساس موارد استفاده واقعی هستند، به شما راه‌حل‌های عملی و آماده برای استفاده در پروژه‌های شما می‌دهند. با ایریس، شما نه تنها یک قالب می‌خرید بلکه یک پایه قدرتمند و انعطاف‌پذیر برای موفقیت توسعه وب خود سرمایه‌گذاری می‌کنید.

کتابخانه‌های هسته استفاده شده

اینجا لیستی از کتابخانه‌های هسته‌ای است که در ایریس استفاده می‌کنیم



تل‌ویند سی‌اس‌اس



تایپ اسکریپت

NEXT.js

نکست.جی‌اس



ری‌اکت

نصب

پیش‌نیازها

قبل از شروع کار، اطمینان حاصل کنید که محیط توسعه شما دارای ابزارهای زیر نصب شده است:

• [Node.js](#)

• [npm](#)

نصب ایریس

بسته شروع‌کننده تنظیمات پایه‌ای برای الگو را فراهم می‌کند. این جایی است که باید توسعه خود را از اینجا شروع کنید، صفحات ایجاد کنید و کد خود را اضافه کنید. ما توصیه می‌کنیم این پوشه را به فضای کار خود کپی کنید.

• [documentation](#)

این پوشه شامل یک فایل `index.html` است که به مستندات آنلاین هدایت می‌کند.

پس از انتخاب بسته مورد نظر خود، به دایرکتوری ریشه پروژه (جایی که `package.json` قرار دارد) بروید و دستور زیر را در کنسول خود اجرا کنید:

```
npm install
```

این کار همه وابستگی‌های لازم را در دایرکتوری node_modules نصب می‌کند، به شما اجازه می‌دهد که توسعه را شروع کنید.

تل‌ویند سی‌اس‌اس

تل‌ویند سی‌اس‌اس یک چارچوب سی‌اس‌اس اولویت‌دار است که کلاس‌های از پیش تعریف شده برای ساخت و طراحی الی مستقیماً در JSX فراهم می‌کند. ایریس از تل‌ویند به عنوان چارچوب سی‌اس‌اس اصلی خود استفاده می‌کند، با بیشتر عناصر الی که کاملاً با استفاده از ویژگی‌های آن ساخته شده‌اند. می‌توانید به راحتی تم و استایل پایه را با اصلاح فایل tailwind.config.cjs در دایرکتوری ریشه به روز رسانی کنید.

ابزارها

اگر از VS Code به عنوان IDE خود استفاده می‌کنید، توصیه می‌کنیم که پلاگین [Tailwind CSS IntelliSense](#) را نصب کنید. این پلاگین خودکامل، برجسته‌سازی نحو و چک کردن بر اساس پیکربندی تل‌ویند شما را فراهم می‌کند، که می‌تواند به طور قابل توجهی فرآیند توسعه شما را سرعت بخشد.

یادداشت

برخی از کامپوننت‌های الی ما از کلاس‌های معنایی با دستور `apply` تل‌ویند استفاده می‌کنند. در برخی موارد، مستقیماً کلاس‌های تل‌ویند را به این کامپوننت‌ها اعمال کردن ممکن است به همان صورت که انتظار می‌رود کار نکند. ممکن است نیاز داشته باشید که از مودیفایر `important!` برای غلبه بر انتخابگرهای پیش‌فرض با اولویت بالا استفاده کنید.

برای اینکه هر کلاس کاربری مهم باشد، فقط یک کاراکتر! در ابتدای آن اضافه کنید، مثلاً:

```
<Dropdown className="!bg-red-500" />
```

برای اطلاعات بیشتر در مورد کلاس‌های کاربری و پیکربندی تل‌ویند، به مستندات رسمی مراجعه

کنید: <https://tailwindcss.com/docs/utility-first>

سیاس اس

در حالی که ما اصولاً از Tailwind CSS استفاده می‌کنیم، ما همچنین استایل‌های اضافی نوشته شده در CSS استاندارد داریم که در دایرکتوری `src/assets/styles/*` قرار دارند. به دلیل اینکه Tailwind به برخی از ویژگی‌های پردازش CSS وابسته است، ما از PostCSS به عنوان پیش‌پردازنده خود استفاده می‌کنیم.

در ادامه، یک مروری بر ساختار پوشه‌های استایل‌های ما داریم:

```
| styles
| | components      # استایل‌های کامپوننت‌های پایه
| | docs            # استایل‌های کامپوننت‌های مستندات
| | tailwind         # Tailwind ورودی و استایل‌های پایه
| | template        # استایل‌های کامپوننت‌های قالب
| | vendors          # استایل‌های کتابخانه‌های شخص ثالث
| | index.css        # اصلی CSS ورودی
```

هر پوشه در دایرکتوری styles شامل یک فایل index.css است که همه فایل‌های CSS دیگر در همان پوشه را وارد می‌کند. در نهایت، همه این index.css فایل‌ها به CSS ورودی اصلی وارد می‌شوند.

استایل CSS سفارشی

اگر Tailwind نیازهای استایل‌دهی شما را کامل نمی‌کند، می‌توانید استایل‌های سفارشی را در این پوشه اضافه کنید. ما توصیه می‌کنیم که هنگام اضافه کردن استایل‌های سفارشی، از توابع و دستورات Tailwind استفاده کنید، مانند `@layer`، `@apply`، و `theme()`.

برای جزئیات بیشتر در مورد استفاده از این ویژگی‌ها، می‌توانید به [مستندات رسمی Tailwind](#) مراجعه کنید.

شروع کننده

همانطور که در بخش [نصب](#) ذکر شد، ما نسخه شروع‌کننده‌ای را فراهم کرده‌ایم که شامل اجزای هسته اساسی و کارکرد پایه است. ما به شدت توصیه می‌کنیم که توسعه‌دهندگان از این نسخه به عنوان پایه برای ساخت اپ‌های خود استفاده کنند.

وقتی بسته شروع‌کننده را در محیط محلی خود باز می‌کنید، به صفحه ورود هدایت می‌شوید. می‌توانید با استفاده از اعتبارنامه‌ها کاربر / *admin*: رمز عبور: *Qwe 123* وارد شوید.

پیکربندی‌های پیش‌فرض

در زیر برخی از پیکربندی‌های پیش‌فرض برای نسخه شروع‌کننده آمده است. می‌توانید این تنظیمات را برای برآوردن نیازهای خود تغییر دهید.

[مستندات](#) AppConfig

```
const appConfig: AppConfig = {
  apiPrefix: '/api',
  authenticatedEntryPath: '/home',
  unAuthenticatedEntryPath: '/sign-in',
  locale: 'en',
  activeNavTranslation: false
}
```

[مستندات](#) ThemeConfig

```
export const themeConfig: ThemeConfig = {
  schema: 'default',
  direction: THEME_ENUM.DIR_LTR,
  mode: THEME_ENUM.MODE_LIGHT,
  panelExpand: false,
  controlSize: 'md',
  layout: {
    type: THEME_ENUM.LAYOUT_COLLAPSIBLE_SIDE,
    sideNavCollapse: false,
  },
},
```

[مستندات](#) RoutesConfig

```
const publicRoutes = [
  '/home': {
    key: 'home',
    authority: [],
  }
]

export const protectedRoutes = {
  '/articles': {
    key: 'articles',
    authority: [ADMIN, USER],
    meta: {
      pageContainerType: 'contained',
    },
  },
  '/articles/[slug]': {
    key: 'articles.articleDetails',
    authority: [ADMIN, USER],
    meta: {
      pageContainerType: 'contained',
    },
    dynamicRoute: true
  },
}
```

NavConfig [مستندات](#)

```
const navigationConfig = [
  {
    key: 'home',
    path: '/home',
    title: 'خانه',
    translateKey: 'nav.home',
    icon: 'home',
    type: NAV_ITEM_TYPE_ITEM,
    authority: [],
    subMenu: []
  }
]
```

تایپ اسکریپت

این راهنما شما را در تنظیمات تایپ اسکریپت و استفاده از ویژگی‌های آن در الستار راهنمایی می‌کند. از آنجا که اکم کاملاً در تایپ اسکریپت توسعه یافته است، به شما اجازه می‌دهد که کد ایمن از نظر نوع و مقیاس‌پذیر بنویسید، که ساخت برنامه‌های وب مدرن را آسان‌تر می‌کند.

تنظیمات تایپ اسکریپت را پیکربندی کنید

پیکربندی مناسب در پروژه‌های تایپ‌اسکریپت ضروری است زیرا به شما اجازه می‌دهد که گزینه‌های کامپایلر مختلف را برای تطابق با نیازهای پروژه خود سفارشی کنید. می‌توانید این تنظیمات را در فایل tsconfig.json انجام دهید.

سختی تایپ‌اسکریپت را تنظیم کنید

ما می‌دانیم که ممکن است همه با چک کردن نوع سخت تایپ‌اسکریپت راحت نباشند. اگر تجربه‌ای شبیه به جاوااسکریپت می‌خواهید، می‌توانید تنظیمات را برای اجازه دادن به چک کردن نوع ضعیف‌تر تنظیم کنید.

اینجا یک مثال از پیکربندی tsconfig.json برای چک کردن نوع ضعیف است:

```
{
  "compilerOptions": {
    "strict": false,
    ...
  }
}
```

این تنظیمات سختی تایپ‌اسکریپت را کاهش می‌دهد، به شما انعطاف‌پذیری بیشتری می‌دهد و تجربه‌ای شبیه به جاوااسکریپت را ارائه می‌کند.

چک تایپ‌اسکریپت را برای ساخت تولید نکس‌جی خاموش کنید

به طور پیش‌فرض، نکس‌جی چک کردن نوع سخت را در طی فرایند ساخت انجام می‌دهد. اگر می‌خواهید چک کردن نوع را برای ساخت تولید خاموش کنید، می‌توانید فایل next.config.js را به صورت زیر تنظیم کنید.

```
module.exports = {
  typescript: {
    // !! هشدار !!
    // به خطر اجازه می‌دهد که ساخت‌های تولید حتی اگر پروژه شما دارای خطاهای نوع باشد، با موفقیت به پایان برسد.
    // !! هشدار !!
    ignoreBuildErrors: true,
  },
}
```

این تنظیمات سختی تایپ‌اسکریپت را کاهش می‌دهد، به شما انعطاف‌پذیری بیشتری می‌دهد و تجربه‌ای شبیه به جاوااسکریپت را ارائه می‌کند.

به روزرسانی

زیرا پلتفرم شامل یک سیستم کنترل نسخه داخلی ندارد، به روزرسانی قالب‌های شما ممکن است چالش برانگیز باشد. با این حال، اینجا چند نکته است که به شما کمک می‌کند تا قالب‌های خود را به روز نگه دارید.

- قبل از شروع یک پروژه جدید، در نظر بگیرید که قالب اصلی را در مخزن خود میزبانی کنید.
- یک شاخه از مخزن ایجاد کنید تا به عنوان فضای کاری خود استفاده کنید.
- هنگامی که یک به‌روزرسانی جدید منتشر می‌شود، محتوای آخرین نسخه را به مخزن خود بفرستید.
- شاخه فضای کاری خود را با شاخه اصلی ادغام کنید تا آخرین تغییرات را دریافت کنید

سرور توسعه

یکبار که همه وابستگی‌ها را نصب کردید، می‌توانید سرور توسعه را با اجرای دستور زیر در ترمینال خود شروع کنید:

```
npm run dev
```

مرورگر خود را باز کنید و به <http://localhost:3000/> بروید. برنامه به‌طور خودکار هر زمان که تغییراتی در فایل‌های منبع ایجاد کنید، مجدداً بارگذاری می‌شود.

متغیرهای محیطی

متغیرهای محیطی راهی امن برای ذخیره و مدیریت داده‌های پیکربندی یا اطلاعات حساس، مانند کلیدهای API، رشته‌های اتصال به پایگاه داده، یا تنظیمات خاص محیط است. در Next.js، متغیرهای محیطی در فایل‌های `.env` تعریف می‌شوند و بسته به پیشوند آنها، در زمان ساخت یا زمان اجرا قابل دسترسی هستند.

تنظیم متغیرهای محیطی

الگو از قبل با یک فایل `.env` در دایرکتوری ریشه همراه است. آزاد هستید این متغیرها را بر اساس نیازمندی‌های پروژه خود تغییر دهید. اینجا یک مثال از متغیرهای محیطی و کنوانسیون نامگذاری است:

```
NEXT_PUBLIC_API_BASE_URL=https://api.example.com
DATABASE_URL=postgres://user:password@localhost:5432/db_name
```

متغیرهای عمومی: متغیرهایی که با `NEXT_PUBLIC_` پیشوند می‌شوند، به مرورگر نمایش داده می‌شوند. اینها نباید شامل داده‌های حساس باشند.

- متغیرهای خصوصی: متغیرهایی که بدون پیشوند `NEXT_PUBLIC_` هستند، فقط در سمت سرور در دسترس هستند و هرگز به کلاینت نمایش داده نمی‌شوند.

دسترسی به متغیرهای محیطی

دسترسی سمت سرور:

متغیرهای خصوصی را در توابع سمت سرور مانند مسیرهای API یا کامپوننت‌های سرور دسترسی کنید:

```
export async function GET() {
  const databaseUrl = process.env.DATABASE_URL;
  return new Response('Database URL: ' + databaseUrl);
}
```

دسترسی سمت کلاینت:

متغیرهای عمومی را در کامپوننت‌های React خود دسترسی کنید:

```
const apiBaseUrl = process.env.NEXT_PUBLIC_API_BASE_URL;

useEffect(() => {
  console.log('API Base URL:', apiBaseUrl);
}, []);
```

مستندات رسمی

برای جزئیات بیشتر، به مستندات رسمی [متغیرهای محیطی Next.js](#) مراجعه کنید.

ساختار پوشه

ساختار پوشه‌های قالب ما به‌طور نزدیک به کنوانسیون‌های رسمی [ساختار مسیر یاب اپ](#) [Next.js](#) پیروی می‌کند. در این بخش، ساختار پوشه پایه و همه چیزهایی که برای راه‌اندازی قالب نیاز دارید را پیدا خواهید کرد. هر دو نسخه دمو و استارتر ساختار یکسانی دارند، به جز اینکه نسخه استارتر تعداد کمتری فایل و پوشه نسبت به نسخه دمو خواهد داشت، زیرا آنها در نسخه استارتر لازم نیستند.

زیر یک نمودار شماتیک از ساختار دایرکتوری است:

```

├─ messages                                # پیام‌های محلی
├─   └─ en.json                          # محلی JSON فایل
├─   └─ ...                              # محلی JSON سایر فایل‌های
├─ public                                 # منبع استاتیک
├─   └─ img                              # تصاویر
├─   └─ data                             # داده‌های استاتیک
├─   └─ ...                              # سایر فایل‌های استاتیک
├─ src
├─   └─ @types                            # ال‌های نوع که در سراسر قالب به اشتراک گذاشته می‌شوند
├─   └─ ...
├─   └─ app                              # Next.js دایرکتوری اصلی برنامه برای مسیریاب اپ
├─     └─ (auth-pages)                   # گروه مسیرها برای صفحات احراز هویت
├─       └─ ...                          # صفحات احراز هویت
├─       └─ layout.tsx                   # طرح پایه برای صفحات احراز هویت
├─       └─ (protected-pages)            # گروه مسیرها برای صفحات که احراز هویت نیاز دارند
├─         └─ ...                          # صفحات محافظت شده
├─         └─ layout.tsx                 # طرح ریشه برای صفحات محافظت شده
├─         └─ (public-pages)             # گروه مسیرها برای صفحات عمومی
├─           └─ ...                       # صفحات عمومی
├─         └─ api                        # API نقطه پایانی
├─           └─ ...                       # API مسیر و فایل‌های
├─         └─ favicon.ico                 # فایکون
├─         └─ layout.tsx                 # طرح ریشه
├─         └─ not-found.tsx              # صفحه خالی برای 404
├─         └─ page.tsx                   # صفحه ورود
├─       └─ assets                       # منبع استاتیک برنامه
├─         └─ maps                       # متادیتای نقشه
├─         └─ markdown                   # فایل‌های مارکداون
├─         └─ styles                     # جهانی CSS فایل‌های
├─         └─ svg                        # SVG فایل‌های
├─       └─ components                   # کامپوننت‌های عمومی
├─         └─ auth                       # کامپوننت‌های مرتبط با احراز هویت
├─         └─ docs                       # کامپوننت‌های مرتبط با مستندات
├─         └─ layouts                    # کامپوننت‌های طرح
├─         └─ shared                     # اند UI کامپوننت‌های سطح بالا که بر اساس کامپوننت‌های
├─         └─ template                   # ... کامپوننت‌های قالب، مانند هدر، فوتر، ناو، و غیره
├─         └─ ui                        # کامپوننت‌های سطح پایین، مانند دکمه، کشویی، و غیره
├─         └─ view                       # های سطح بالا که برای صفحات به اشتراک گذاشته می‌شوند
├─       └─ configs                      # فایل‌های پیکربندی
├─         └─ ...
├─       └─ constants                    # فایل‌های ثابت
├─         └─ ...
├─       └─ i18n                         # تنظیمات محلی‌سازی
├─         └─ ...
├─       └─ mock                         # جعلی API داده‌های نمونه برای تماس‌های
├─         └─ data                       # داده‌های نمونه
├─         └─ ...                       # داده‌های نمونه TS فایل‌های
├─       └─ server                       # فایل‌های سمت سرور
├─         └─ actions                    # فایل‌های اقدام سمت سرور
├─         └─ ...
├─       └─ services                     # اشتراکی API فایل‌های سرویس برای مدیریت یکپارچه‌سازی‌های
├─         └─ ApiService.ts              # API دستگیره درخواست و پاسخ
├─         └─ axios                      # axios پیکربندی‌ها و اینترسپتورهای
├─           └─ ...
├─         └─ ...                       # سایر فایل‌های سرویس
├─       └─ utils                        # های قابل استفاده مجدد hook همه توابع و
├─         └─ hooks                      # hook‌ها
├─         └─ ...
├─         └─ ...                       # توابع قابل استفاده مجدد
├─         └─ auth.ts                    # Next فایل ورود به سیستم اکسپ
├─         └─ middleware.ts              # Nextjs میان‌افزار
├─       └─ .env                         # فایل برای ذخیره پیکربندی محیط و اسرار
├─       └─ .eslintignore                 # eslint فایل نادیده گرفتن برای
├─       └─ .gitignore                   # git فایل نادیده گرفتن برای
├─       └─ .prettierrc                  # prettier فایل نادیده گرفتن برای
├─       └─ .prettierrc                  # prettier پیکربندی
├─       └─ next-env.d.ts                # Nextjs اعلام محیط
├─       └─ next.config.mjs              # Nextjs فایل پیکربندی
├─       └─ package.json                 #
├─       └─ package.lock.json            #
├─       └─ postcss.config.mjs           # PostCSS فایل پیکربندی
├─       └─ README.md                    #
├─       └─ tailwind.config.ts           # TailwindCSS فایل پیکربندی
├─       └─ tsconfig.json                # Typescript پیکربندی پروژه

```

این ساختار پوشه‌ها سازمان‌دهی واضحی از منابع، کامپوننت‌ها، پیکربندی‌ها، و دارایی‌ها را فراهم می‌کند، که مدیریت و مقیاس‌بندی پروژه شما را آسان‌تر می‌کند. هر پوشه و فایل به‌طور عمدی در جای خود قرار گرفته تا یک پایه کد تمیز و قابل نگهداری را تضمین کند.

مسیریابی

مسیریابی Ecme از کنوانسیون‌های مسیریابی Next.js App Router پیروی می‌کند، از یک سیستم مسیریابی مبتنی بر فایل استفاده می‌کند که هر فایل در دایرکتوری `/app` به یک مسیر در اپلیکیشن شما مربوط می‌شود. زیردایرکتوری‌ها نمایانگر مسیرهای تودرتو هستند، که ساختار کردن ناوبری پیچیده را آسان می‌کند.

برای اطلاعات مفصل و مفاهیم درباره مسیریابی Next.js App Router، لطفاً به [مستندات](#) مراجعه کنید.

پیکربندی مسیر

اگرچه Next.js یک سیستم مسیریابی قدرتمند و قوی از جعبه خارج می‌دهد، اما تمپلیت ما یک لایه اضافی از پیکربندی مسیریابی را معرفی می‌کند. این رویکرد به ما اجازه می‌دهد تا رفتار هر صفحه را در حالی که انعطاف‌پذیری و مقیاس‌پذیری را حفظ می‌کند، گسترش و سفارشی کنیم. پیکربندی مسیریابی برای تمپلیت در `src/configs/routes.config/index.ts` یافت می‌شود. سه گروه اصلی از مسیرها وجود دارد:

```
export const publicRoutes = [
  ...
]

export const protectedRoutes = [
  ...
]

export const authRoutes = [
  ...
]
```

publicRoutes:

این گروه شامل همه مسیرهایی است که برای همه کاربران قابل دسترسی هستند.

protectedRoutes: •

این گروه مسیرهایی را که برای دسترسی به آنها احتیاج به احراز هویت دارند، در بر می‌گیرد.

authRoutes: •

این گروه پیکربندی مسیرهایی را که مربوط به فرآیندهای ورود، ثبت نام و احراز هویت هستند، مدیریت می‌کند.

ساختار پیکربندی

پیکربندی مسیریابی سفارشی ما یک راه ساختاریافته برای تعریف و گسترش رفتار هر مسیر در اپلیکیشن را معرفی می‌کند. اینجا یک تجزیه از ساختار پیکربندی است:

```
export const protectedRoutes = {
  '/articles': {
    key: 'articles',
    authority: [ADMIN, USER],
    meta: {
      pageContainerType: 'contained',
    },
  },
  '/articles/[slug]': {
    key: 'articles.articleDetails',
    authority: [ADMIN, USER],
    meta: {
      pageContainerType: 'contained',
    },
    dynamicRoute: true
  },
}
```

اجزای اصلی پیکربندی:

1. کلیدهای مسیر (مچرها):

هر کلید در شیء (routes برای مثال، ('/articles','/articles/[slug]') به عنوان یک مچر مسیر برای URL فعلی عمل می‌کند. هنگامی که کاربر به یک صفحه می‌رود، اپلیکیشن

URL را با کلیدهای مسیر در پیکربندی مقایسه می‌کند. یکبار که یک تطابق پیدا شد، داده‌های مسیر مربوطه را از شیء پیکربندی بازیابی می‌کند و تنظیمات تعریف شده را اعمال می‌کند.

برای مثال:

- 'articles/' یک مسیر استاتیک را تطابق می‌دهد.
- 'articles/[slug]/' یک مسیر دینامیک را تطابق می‌دهد، جایی که [slug] جایگزین با مقدار دینامیک واقعی از URL می‌شود.

2. شیء پیکربندی:

هر کلید مسیر به یک شیء پیکربندی حاوی خواص زیر نقشه می‌شود:

- key: یک شناساگر برای مسیر که با پیکربندی ناوبری جفت می‌شود.
- authority: یک آرایه از نقش‌ها (برای مثال، (ADMIN, USER)) که مشخص می‌کند کدام نقش‌های کاربر اجازه دسترسی به مسیر را دارند. این کنترل دسترسی بر اساس نقش (RBAC) را پیاده‌سازی می‌کند.
- meta: متادیتا برای مسیر، گزینه‌های سفارشی بیشتر را فراهم می‌کند:
- dynamicRoute: یک پرچم بولین که نشان می‌دهد آیا مسیر دینامیک است (یعنی حاوی جایگزین پارامتر [slug] است).

اختیار

مسیرهای Ecme از کنترل دسترسی ساده بر اساس نقش پشتیبانی می‌کنند. می‌توانید نقش‌هایی را که به یک مسیر دسترسی دارند، با استفاده از فیلد authority مشخص کنید. برای مثال، مسیر زیر فقط برای کاربران با نقش‌های 'admin' یا 'user' قابل دسترسی است. اگر فیلد authority به عنوان یک آرایه خالی باقی بماند، مسیر برای همه نقش‌ها باز خواهد بود.

```
export const protectedRoutes = [
  {
    ...
    authority: ['admin', 'user'],
  },
]
```

البته مدل اختیار انعطاف‌پذیر است و می‌تواند برای برآوردن الزامات خاص پروژه تغییر کند. می‌توانید آن را برای دسترسی کاربر در سطح میانی سرور یا مستقیماً در سطح کلاینت برای کنترل بیشتر پیکربندی کنید.

متا

فیلد `meta` به شما اجازه می‌دهد تا اطلاعات اضافی را به `PageContainer` یا کامپوننت نمایشی مرتبط با مسیر منتقل کنید.

```
export const protectedRoutes = [
  {
    ...
    meta: {
      pageContainerType: 'gutter',
      header: {
        title: 'My tittle',
        description: 'Some description'
        contained: false,
        extraHeader: lazy(() => import('@components/SomeComponent')),
      },
      footer: false,
      layout: 'blank',
    },
  },
]
```

چیدمان قادر خواهد بود به همه داده‌های `meta` مشخص شده دسترسی داشته باشد.

ایجاد صفحه جدید

این راهنما شما را از طریق مراحل ایجاد یک صفحه جدید در قالب راهنمایی می‌کند. با استفاده از مسیریابی سیستم فایل Next.js و پیکربندی مسیریابی گسترده، می‌توانید به سادگی صفحات جدید را با تنظیمات سفارشی اضافه کنید.

ایجاد کامپوننت صفحه

1. به دایرکتوری اپ ناوبری کنید

در پوشه `src/app` تصمیم بگیرید که صفحه جدید شما کجا باید قرار گیرد. برای مثال، اگر صفحه نیاز به احراز هویت دارد، آن را در دایرکتوری `src/app/(protected-pages)` قرار دهید. اگر احراز هویت لازم ندارد، از دایرکتوری `src/app/(public-pages)` استفاده کنید.

2. فایل جدید ایجاد کنید

فایلی با نام مسیر جدید ایجاد کنید. برای مثال:

- برای یک مسیر استاتیک، `src/app/new-page/page.tsx` ایجاد کنید.
- برای یک مسیر دینامیک، از براکت‌ها برای تعریف پارامترها استفاده کنید، مانند `src/app/new-page/[id]/page.tsx`

3. کد صفحه خود را اضافه کنید

کامپوننت ریکت برای صفحه خود را تعریف کنید. مثال:

```
const Page = () => {  
  return (  
    <div>  
      <h1>به صفحه جدید خوش آمدید</h1>  
      <p>این یک صفحه سفارشی است</p>  
    </div>  
  );  
};  
  
export default Page;
```

پیکربندی مسیریابی اضافه کنید

پیکربندی مسیریابی را گسترش دهید،

فایل `src/configs/route.config/routes.config.ts` `protectedRoutes`

یا `publicRoutes` را برای شامل کردن صفحه جدید خود به روز رسانی کنید. برای مثال:

```
import { ADMIN, USER } from '@constants/roles';

export const protectedRoutes = {
  ...protectedRoutes,
  '/new-page': {
    key: 'newPage',
    authority: [ADMIN, USER],
    meta: {
      pageContainerType: 'contained',
    },
  },
};
```

در `Next.js`، ایجاد یک صفحه جدید به سادگی اضافه کردن یک دایرکتوری و یک فایل `page.tsx` است

یادداشت‌های اضافی

به جای اعمال `'use client'` مستقیماً به فایل صفحه، توصیه می‌شود که یک کامپوننت کلاینت جداگانه ایجاد کنید و آن را به صفحه وارد کنید. این رویکرد اجازه می‌دهد که صفحه خود به صورت رندر شده در سرور باقی بماند، رندر سرور (SSR) را بهینه می‌کند و فقط منطق سمت

کلاينت را در کامپوننت کلاينت مدیریت می‌کند. برای مثال:

```
// src/app/(protected-pages)/new-page/page.tsx
import ClientComponent from './ClientComponent';

const NewPage = () => {
  return (
    <div>
      <h1>صفحه جدید</h1>
      <ClientComponent />
    </div>
  );
};

export default NewPage;
```

```
// src/app/(protected-pages)/new-page/_components/ClientComponent.tsx
'use client';

const ClientComponent = () => {
  return <p>این کامپوننت در کلاينت اجرا می‌شود</p>;
};

export default ClientComponent;
```

اینگونه، صفحه می‌تواند قبل از رندر کردن کامپوننت‌های سمت کلاينت، کارهای SSR را انجام دهد.

یکپارچه‌سازی API

Next.js یک سیستم مسیریابی API داخلی دارد که به شما امکان می‌دهد تا قابلیت‌های پشتیبانی را مستقیماً در برنامه خود تعریف کنید. این سیستم به شما امکان مدیریت درخواست‌ها، پردازش داده‌ها و ارتباط با سرویس‌های خارجی را به طور بی‌درز فراهم می‌کند، همه در حالی که منطق API خود را با ساختار پروژه Next.js خود همراه می‌کنید. در این بخش، ما به بررسی نحوه کار API ها در Next.js می‌پردازیم، مثال‌هایی از ایجاد مسیرهای API ارائه می‌دهیم و نشان می‌دهیم چگونه با Ecme کار می‌کند.

نحوه کار مسیرهای API

1. مسیرهای API مبتنی بر فایل:

مسیرهای API در داخل دایرکتوری `/app/api` ایجاد می‌شوند. هر فایل در این دایرکتوری

یک نقطه پایانی تعریف می کند. نام فایل (یا ساختار پوشه) با مسیر پایانی نقطه مطابقت دارد.

2. اشیاء درخواست و پاسخ:

مسیرهای API از طریق اشیاء `req` و `res` سبک `Node.js` درخواست ها و پاسخ ها را مدیریت می کنند. این باعث می شود که پردازش داده های ورودی و ارسال پاسخ ها آسان شود.

3. اجرای سمت سرور:

مسیرهای API همیشه در سرور اجرا می شوند، اطمینان از اینکه اجرای امور حساس مانند پرس و جوی پایگاه داده یا منطق احراز هویت امن انجام می شود.

برای جزئیات جامع تر، به [مستندات Next.js در مورد مسیرهای API](#) مراجعه کنید.

ایجاد مسیرهای API

در اینجا یک مثال از ایجاد یک مسیر API ساده آورده شده است

1. ایجاد مسیر API:

یک فایل به نام `hello.ts` در دایرکتوری `src/app/api/hello/route.ts` ایجاد کنید:

```
import { NextResponse } from 'next/server';

const mockMethodThatSavesToDatabase = async (data) => {
  // شبیه سازی ذخیره سازی در پایگاه داده
  console.log('داده در پایگاه داده ذخیره شد', data);
};

export async function POST(request: Request) {
  try {
    // خارجی دریافت کنید API داده ها را از یک
    const externalApiResponse = await fetch('https://jsonplaceholder.typicode.com/posts/1');
    const externalData = await externalApiResponse.json();

    // داده های دریافتی را در پایگاه داده ذخیره کنید
    await mockMethodThatSavesToDatabase(externalData);

    return NextResponse.json({ message: 'داده با موفقیت ذخیره شد', data: externalData });
  } catch (error) {
    return NextResponse.json({ error: 'ناتوان در ذخیره داده', details: error.message }, { status: 500 });
  }
}
```

فراخوانی یک مسیر API از سمت مشتری

برای ساده سازی فراخوانی های API از سمت مشتری، قالب ما شامل یک ابزار ApiService زیر دایرکتوری services را شامل می شود. این ابزار: پیکربندی پذیرفتن پیکربندی درخواست Axios و به طور خودکار پاسخ ها و خطاها را مدیریت می کند، فقط داده های مربوط یا جزئیات خطا را برمی گرداند. اینجا است چگونه کار می کند:

1. ابتدا یک فایل سرویس جدید برای ویژگی یا ماژول خود ایجاد کنید. به عنوان مثال، اگر در مدیریت کاربران کار می کنید، یک فایل به نام UserManagementService.ts در دایرکتوری services ایجاد کنید.

2. در داخل فایل سرویس، یک تابع ناهمزمان برای مدیریت درخواست API تعریف کنید. این تابع باید از ApiService.fetchData استفاده کند، دو نوع جنریک را پذیرفته و تنظیمات Axios را پذیرفته Response و Request، همراه با داده. اینجا یک مثال است:

```
type MyApiResponse = {
  someResponseData: string
  someResponseData2: boolean
}

type MyApiRequest = {
  someRequestData: string
}

export async function myApi (data) {
  return ApiService.fetchData<MyApiResponse,MyApiRequest>({
    url: '/my-api-url',
    method: 'post',
    data
  })
}
...
```

یا انتقال نوع به صورت جنریک از سطح مصرف کننده:

```
import ApiService from "../ApiService"

export async function myApi<TResponse, TRequest>(data) {
  return ApiService.fetchData<TResponse, TRequest>({
    url: '/my-api-url',
    method: 'post',
    data
  })
}
...
```

و اکنون می توانید این API را در کامپوننت خود متصل کنید

```
type MyApiResponse = {
  someResponseData: string
  someResponseData2: boolean
}

type MyApiRequest = {
  someRequestData: string
}

import { myApi } from '../MyService.ts'

const MyComponent = props => {

  const fetchData = async () => {
    const requestParam = { key: 'value' }
    try {
      const resp = await myApi<MyApiResponse, MyApiRequest>(requestParam)
      if (resp.data) {
        ...do something
      }
    } catch (errors) {
      ...handle errors
    }
  }

  // You can
  useEffect(() => {
    fetchData()
  }, [])

  return (
    ...
  )
}
```

توجه: همچنین می توانید از کتابخانه های دریافت داده مانند [SWR](#) یا [TanStack Query](#) برای رویکرد اظهاری به دریافت داده استفاده کنید. انتخاب بستگی به نیازهای خاص شماست.

با ApiService، مدیریت فراخوانی های API از سمت مشتری به یک استاندارد تبدیل می شود و آسان تر برای نگهداری می شود. با مسیرهای API Next.js ترکیب شده، می توانید منطق سمت سرور را به طور بی درز مدیریت کرده و با پشتیبانی خود ارتباط برقرار کنید.

احراز هویت

Ecme از NextAuth.js به عنوان راه حل احراز هویت استفاده می کند، که یک راه حل قوی و انعطاف پذیر برای مدیریت احراز هویت کاربر ارائه می دهد، از جریان های احراز هویت سفارشی پشتیبانی می کند و طیف گسترده ای از ارائه دهندگان احراز هویت، از جمله گوگل، گیت هاب، و بیشتر را ارائه می دهد.

NextAuth به متغیرهای محیطی برای پیکربندی امن وابسته است. به طور پیش فرض، الگوی ما شامل متغیرهای زیر در فایل .env است:

```
# کلید مخفی احراز هویت
AUTH_SECRET=کلید-مخفی-تو

# URL پایه برای فراخوانی های NextAuth
NEXTAUTH_URL=http://localhost:3000

# ارائه دهنده احراز هویت (گوگل)
کلید-مشتری-گوگل=تو=GOOGLE_AUTH_CLIENT_ID
کلید-مخفی-گوگل=تو=GOOGLE_AUTH_CLIENT_SECRET

# ارائه دهنده احراز هویت (گیت هاب)
کلید-مشتری-گیت-هاب=تو=GIT_HUB_AUTH_CLIENT_ID
کلید-مخفی-گیت-هاب=تو=GIT_HUB_AUTH_CLIENT_SECRET
```

AUTH_SECRET: یک رشته تصادفی برای رمزگذاری توکن های جلسه استفاده می شود.

- **NEXTAUTH_URL:** URL پایه برای اپلیکیشن شما، در فراخوانی های بازگشتی استفاده می شود. این را در تولید به روز رسانی کنید تا با دامنه مستقر شده شما مطابقت داشته باشد.

- **GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET, GITHUB_CLIENT_ID, GITH**
UB_CLIENT_SECRET, : اعتبارنامه ها برای یکپارچه سازی با OAuth.

در الگوی ما، پیکربندی NextAuth در فایل `auth.config.ts` متمرکز شده است که در `src/configs/` قرار دارد. این فایل ارائه دهندگان احراز هویت و دستگیره های فراخوانی را برای گسترش ویژگی های جلسه تعریف می کند.

ارائه دهنده اعتبارنامه

ارائه دهنده Credentials اجازه می دهد تا منطق احراز هویت سفارشی را از طریق روش `authorize` انجام دهید، هنگامی که کاربر وارد می شود، اعتبارنامه ها با استفاده از یک اقدام پیشی `validateCredential` اعتبار سنجی می شوند. اگر اعتبار سنجی موفق باشد، شیء کاربر بازگردانده می شود؛ در غیر این صورت، `null` بازگردانده می شود. می توانید منطق اعتبار سنجی کاربر خود را در روش `validateCredential` پیاده سازی کنید، می تواند یک فراخوانی API یا یک پرس و جو در پایگاه داده باشد.

```
import type { NextAuthConfig } from "next-auth";
import validateCredential from "../server/actions/user/validateCredential";
import Credentials from "next-auth/providers/credentials";
import Github from "next-auth/providers/github";
import Google from "next-auth/providers/google";

import type { SignInCredential } from "@types/auth";

export default {
  providers: [
    ..., // دیگر ارائه دهندگان، مثلاً گیت هاب، گوگل
    Credentials({
      async authorize(credentials) {
        /** اعتبارنامه ها را از یک اند اینچ اعتبار سنجی کنید */
        const user = await validateCredential(credentials as SignInCredential)
        if(!user) {
          return null
        }

        return {
          id: user.id,
          name: user.userName,
          email: user.email,
          image: user.avatar,
        };
      },
    }),
  ],
}
```

ادامه 

```

callbacks: {
  async session(payload) {
    /** بخش اضافه می کنیم 'id' & 'authority' ویژگی های کاربر اضافی را اینجا اعمال کنید، برای مثال، ما
    return {
      ...payload.session,
      user: {
        ...payload.session.user,
        id: payload.token.sub,
        authority: ['admin', 'user']
      },
    },
  },
},
} satisfies NextAuthConfig;

```

یکپارچه سازی OAuth

NextAuth از طیف گسترده ای از ارائه دهندگان OAuth پشتیبانی می کند، می توانید این لینک را برای ارائه دهندگان داخلی بیشتر NextAuth بررسی کنید.

```

export default {
  providers: [
    Google({
      clientId: process.env.GOOGLE_AUTH_CLIENT_ID,
      clientSecret: process.env.GOOGLE_AUTH_CLIENT_SECRET,
    }),
  ],
}

```

دسترسی به جلسات در سمت کلاینت و سرور

احراز هویت اغلب مدیریت جلسه را در هر دو محیط کلاینت و سرور نیاز دارد. الگوی ما ابزارهایی برای مدیریت جلسات به صورت هماهنگ در هر دو محیط فراهم می کند.

• سمت کلاینت:

در حالی که hook useSession NextAuth را فراهم می کند، اما مشکلات بازنویسی شناخته شده دارد. برای حل این مشکل، ما hook جلسه سفارشی به نام useCurrentSession را اضافه کرده ایم. این hook به همان صورت اما با نقص کمتر برای مدیریت وضعیت جلسه در سمت کلاینت استفاده می کند.

```
import useCurrentSession from '@/utils/hooks/useCurrentSession';

const Component = () => {
  const { session } = useCurrentSession();

  return (
    <div>
      {session ? (
        <>
          <h1>خوش آمدید، {session.user.name}!</h1>
          <p>ایمیل: {session.user.email}</p>
        </>
      ) : (
        <p>شما وارد نشده اید.</p>
      )}
    </div>
  );
};

export default Component
```

سمت سرور:

برای عملیات سمت سرور، می توانید از تابع `auth()` برای بازیابی جزئیات جلسه به صورت امن استفاده کنید. این `özellikle` زمانی مفید است که شما دسترسی کاربر به برخی مسیرها را محدود می کنید یا قبل از رندر صفحه داده های کاربر را پیش بارگذاری می کنید.

```
import { auth } from '@auth';

export default async function Page() {
  const session = await auth();

  if (!session) {
    // برای کاربران غیرمجاز، بازنویسی یا نمایش خطا
    return <p>برای مشاهده این صفحه باید وارد شده باشید</p>;
  }

  return (
    <div>
      <h1>خوش آمدید، {session.user.name}!</h1>
      <p>ایمیل شما: {session.user.email}</p>
    </div>
  );
}
```

حذف احراز هویت

اگر تصمیم گرفتید از NextAuth استفاده نکنید، برای حذف آن از این مراحل پیروی کنید:

1. NextAuth را حذف کنید:

```
npm uninstall next-auth
```

فایل های مربوط به NextAuth را حذف کنید:

- فایل `src/configs/auth.config.ts` را حذف کنید.
- مسیرهای API پویا در `pages/api/auth/[...nextauth]` را حذف کنید.
- فایل `middleware.ts` را بازنشانی یا حذف کنید (اگر محافظت مسیر لازم نیست).
- همه پیاده سازی های `useCurrentSession()` & `auth()` را از کامپوننت های زیر حذف کنید:

- `app/layout.tsx`
- `components/template/HorizontalNav.tsx`
- `components/template/MobileNav.tsx`
- `components/template/SideNav.tsx`
- `components/template/UserProfileDropdown.tsx`
- `components/template/StackeSideNav.tsx`

با پیروی از این مراحل، می توانید NextAuth را از الگوی خود حذف کنید و با روش احراز هویت مورد نظر خود جایگزین کنید.

مستندات رسمی

برای جزئیات بیشتر در مورد استفاده از NextAuth، به مستندات `NextAuth.js` مراجعه کنید.

مدیریت وضعیت

در این الگو، Zustand را برای مدیریت حالت سمت کلاینت یکپارچه کرده‌ایم Zustand. یک راه حل مدیریت حالت کوچک، سریع، و مقیاس‌پذیر است که به شما اجازه می‌دهد تا حالت‌های **компонент**‌های سمت کلاینت خود را به شیوه‌ای ساده و کارآمد مدیریت کنید.

ما از Zustand برای مدیریت برخی از حالت‌های هسته‌ای در دمو استفاده می‌کنیم، اما کاملاً اختیاری است که شما در پروژه‌های خود از آن استفاده کنید. اگر ترجیح می‌دهید از کتابخانه مدیریت حالت دیگری یا حتی حالت داخلی ری‌اکت استفاده کنید، می‌توانید به راحتی به آنچه که بیشترین نیاز شما را برآورده می‌کند، تغییر دهید.

ایجاد حالت Zustand

ایجاد یک فروشگاه Zustand ساده است. زیر یک مثال از چگونگی ایجاد یک حالت جهانی برای مدیریت یک شمارنده ساده است:

```
import create from 'zustand'

type CounterState = {
  count: number
}

type CounterAction = {
  increaseCount: () => void
  decreaseCount: () => void
}

// فروشگاه را تعریف کنید
const useCounterStore = create<CounterState & CounterAction>((set) => ({
  count: 0,
  increaseCount: () => set((state) => ({ count: state.count + 1 })),
  decreaseCount: () => set((state) => ({ count: state.count - 1 })),
}))

export default useCounterStore
```

در این مثال، ما یک فروشگاه با یک حالت **count** و دو عمل **increaseCount** و **decreaseCount** ایجاد کرده‌ایم. این اعمال حالت را با افزایش یا کاهش **count** به روز می‌کنند.

استفاده از حالت Zustand در یک کامپوننت

یکبار که فروشگاه را ایجاد کرده‌اید، استفاده از آن در یک کامپوننت ساده است. اینجا چگونگی یکپارچه کردن `useCounterStore` در یک کامپوننت ری‌اکت است:

```
import useCounterStore from './path/to/store'

const Counter = () => {
  const { count, increaseCount, decreaseCount } = useCounterStore()

  return (
    <div>
      <h1>شمارنده: {count}</h1>
      <button onClick={increaseCount}>افزایش</button>
      <button onClick={decreaseCount}>کاهش</button>
    </div>
  )
}

export default Counter
```

در این کامپوننت، ما از hook `useCounterStore` برای دسترسی به حالت `count` و دو عمل استفاده می‌کنیم. به‌طور خودکار هر زمان که حالت تغییر کند، به روز می‌شود، و دکمه‌ها به کاربر اجازه می‌دهند تا با حالت تعامل کنند.

این فقط یک مثال پایه‌ای است تا شما را شروع کند. Zustand انعطاف‌پذیر است و می‌تواند برای سناریوهای مدیریت حالت پیچیده‌تر به `medida` که برنامه شما رشد می‌کند، استفاده شود. اگر می‌خواهید استفاده پیشرفته‌تر را کشف کنید، ما توصیه می‌کنیم که مستندات رسمی [Zustand](#) را بررسی کنید.

پیکربندی اپ

می‌توانید تنظیمات مختلف اپلیکیشن استاتیک را در `src/configs/app.config.ts` پیکربندی کنید. زیر پیکربندی‌های پیش‌فرض برای هر دو نسخه دمو و استارتر هستند.

```
const appConfig: AppConfig = {
  apiPrefix: '/api',
  authenticatedEntryPath: '/dashboards/ecommerce',
  unAuthenticatedEntryPath: '/sign-in',
  locale: 'en',
  activeNavTranslation: true
}
```

```
const appConfig: AppConfig = {
  apiPrefix: '/api',
  authenticatedEntryPath: '/home',
  unAuthenticatedEntryPath: '/sign-in',
  locale: 'en',
  activeNavTranslation: false
}
```

چیدمان‌ها

Ecme شش نوع چیدمان پس از ورود و سه نوع چیدمان احراز هویت ارائه می‌دهد، همه کامپوننت‌های چیدمان می‌توانند در دایرکتوری `src/components/layouts/PostLoginLayout/components*` یافت شوند و همه کامپوننت‌هایی که در چیدمان‌ها استفاده می‌شوند می‌توانند در `src/components/template/*` یافت شوند.

چیدمان‌های پس از ورود که داشتیم عبارت بودند از:

- کناری جمع شونده 'collapsibleSide'
- کناری چیده شده 'stackedSide'
- نوار بالای کلاسیک 'topBarClassic'
- کناری بدون قاب 'framelessSide'
- پوشش محتوا 'contentOverlay'
- خالی 'blank'

پیکربندی چیدمان

می‌توانید چیدمان اولیه را در `src/configs/theme.config.ts` با مقدار رشته‌ای بالا پیکربندی کنید

```
export const themeConfig = {
  ...
  layout: {
    type: 'framelessSide',
    ...
  },
}
```

اینجا مقادیر و کلید در دسترس برای پیکربندی فیلد `layout`

بازنویسی چیدمان‌ها

به طور کلی، همه نمایش‌های مسیر از تنظیمات چیدمان در `theme.config.ts` پیروی می‌کنند. با این حال، اگر مواردی وجود داشته باشد که می‌خواهید در یک مسیر نمایش خاص چیدمانی متفاوت نمایش دهید، می‌توانید مقدار چیدمانی که می‌خواهید در `meta` مسیر به جای چیدمان فعلی بازنویسی کنید، همانطور که در راهنمای مسیریابی ذکر شد.

```
export const protectedRoutes = {
  '/your-page-path': {
    key: 'keyForYourPage',
    authority: [ADMIN, USER],
    meta: {
      ...
      layout: 'blank',
    },
  },
}
```

چیدمان‌های احراز هویت

پیکربندی چیدمان احراز هویت کمی متفاوت از بالا است، فقط باید به `src/app/(auth- pages)/layout.tsx` مراجعه کنید و کامپوننت پوشش را جایگزین کنید. برای مثال، از `side` به `simple` تغییر دهید

```
1 import { ReactNode } from 'react'
2 import Side from '@components/layouts/AuthLayout/Side'
3 import Simple from '@components/layouts/AuthLayout/Simple'
4 // import Split from '@components/layouts/AuthLayout/Split'
5
6 const Layout = ({ children }: { children: ReactNode }) => {
7   return (
8     <div className="flex flex-auto flex-col h-[100vh]">
9       <Simple>
10         <Side>
11           {children}
12         </Side>
13       </Simple>
14     </div>
15   )
16 }
17
18 export default Layout
```

پیکربندی ناوبری

ما ساختار ناوبری خود را به عنوان آرایه ای از اشیاء تشکیل می دهیم و در نهایت آن را به `ala` تبدیل می کنیم. شما می توانید به راحتی ساختار ناوبری برنامه را با دسترسی به `src/configs/navigation.config/index.ts` تغییر دهید یا سفارشی سازی کنید

اینجا نوع یک آیتم منو را مشاهده می کنید

```
export type HorizontalMenuMeta = {
  layout: 'default'
} | {
  layout: 'columns'
  showColumnTitle?: boolean
  columns: 1 | 2 | 3 | 4 | 5
} | {
  layout: 'tabs'
  columns: 1 | 2 | 3 | 4 | 5
}

export interface NavigationTree {
  key: string
  path: string
  isExternalLink?: boolean
  title: string
  translateKey: string
  icon: string
  type: 'title' | 'collapse' | 'item'
  authority: string[]
  subMenu: NavigationTree[]
  description?: string
  meta?: {
    horizontalMenu?: HorizontalMenuMeta
    description?: {
      translateKey: string
      label: string
    }
  }
}
```

تنظیم آیکون ناوبری

تنظیمات آیکون ناوبری در یک فایل جداگانه در

src/configs/navigation-icon.config.tsx قرار دارد

در مثال بالا، ما از مقدار رشته ای uiComponents در فیلد icon استفاده می کنیم، باید سپس از این مقدار در navigation-icon.config.tsx استفاده کنیم تا آیکون را برای تماس بازگویی کنیم.

ابتدا، آیکون مورد نظر خود را از [react-icons](#) وارد کنید

```
import { FaBeer } from 'react-icons/fa'

const navigationIcon = {}
```

مقدار استفاده شده در `icon` را تنظیم کنید و آیکون وارد شده را به عنوان مقدار

```
import { FaBeer } from 'react-icons/fa'

const navigationIcon = {
  uiComponents: <FaBeer />
}
```

اکنون آیتم منو مربوطه `FaBeer` را به عنوان آیکون منو بازگویی می کند.

ساخت تولید

این بخش مراحل استقرار اپلیکیشن شما در محیط زنده را پوشش می دهد. چه از Vercel استفاده می کنید یا راه حل میزبانی سفارشی، برای اطمینان از فرآیند استقرار صاف، از این دستورالعمل ها پیروی کنید. برای دستورالعمل های خاص پلتفرم برای استقرارها، به [مستندات استقرار Next.js](#) مراجعه کنید.

آماده سازی برای استقرار

قبل از استقرار، اطمینان حاصل کنید که اپلیکیشن شما آماده تولید است:

- متغیرهای محیط: تأیید کنید که همه متغیرهای مورد نیاز (مثل کلیدهای API ، URL های دیتابیس) در پیکربندی محیط شما به درستی تنظیم شده اند.
- بهینه سازی های ساخت

برای ایجاد یک ساخت تولید بهینه، دستور زیر را اجرا کنید:

```
npm run build
```

تست کردن

اپلیکیشن خود را با استفاده از ساخت تولید به صورت محلی تست کنید تا مسائل احتمالی را شناسایی کنید:

```
npm run start
```

استقرار به Vercel

Vercel یک پلتفرم محبوب برای استقرار اپلیکیشن‌های Next.js است.

1. اتصال به Vercel: به [Vercel](#) وارد شوید و مخزن Git خود را وارد کنید.
2. تنظیمات ساخت: اطمینان حاصل کنید که چارچوب به Next.js تنظیم شده است، و دستور ساخت `npm run build` است.
3. متغیرهای محیط: متغیرهای محیط را مستقیماً در داشبورد Vercel اضافه کنید.
4. استقرار: روی استقرار کلیک کنید، و Vercel بقیه کارها را انجام خواهد داد.

میزبانی سفارشی

اگر به سرور یا VPS سفارشی استقرار می‌دهید، اطمینان حاصل کنید که Node.js در سرور شما نصب شده است.

۱. ساخت اپلیکیشن: ساخت تولید را اجرا کنید

```
npm run build
```

۲. ساخت اپلیکیشن

```
npm run start
```

می‌توانید از یک مدیر فرایند مانند `pm2` یا `Docker` برای مدیریت اپلیکیشن خود استفاده کنید