جزئیات زیرساخت سامانه سپهتن

**فهرست مطالب**

[**نمای کلی** 3](#_Toc183526321)

[**اهداف پروژه** 3](#_Toc183526322)

[**معماری سیستم در بخش فرانت‌اند** 3](#_Toc183526323)

[**لایه‌بندی کامپوننت‌ها** 3](#_Toc183526324)

[**ساختار پوشه‌بندی مبتنی بر ویژگی** 4](#_Toc183526325)

[**ساختار پوشه‌ها** 4](#_Toc183526326)

[**توضیحات بیشتر** 4](#_Toc183526327)

[**کتابخانه‌ها و ابزارها** 6](#_Toc183526328)

[**کتابخانه‌های فرانت‌اند** 6](#_Toc183526329)

[**استایل‌دهی** 6](#_Toc183526330)

[**تست** 6](#_Toc183526331)

[**گردش‌کار توسعه** 7](#_Toc183526332)

[**فرآیند توسعه ویژگی** 7](#_Toc183526333)

[**کنترل نسخه** 7](#_Toc183526334)

[**تست** 7](#_Toc183526335)

[**استانداردهای کدنویسی و بهترین روش‌ها** 7](#_Toc183526336)

[**بهبودهای آینده** 7](#_Toc183526337)

# **نمای کلی**

این مستند ساختار معماری، سیستم لایه‌بندی کامپوننت‌ها، و فناوری‌های استفاده‌شده برای ساخت داشبورد مدیریت ناوگان را شرح می‌دهد. هدف این مستند، ارائه راهنمایی برای درک، نگهداری و گسترش اپلیکیشن است.

# **اهداف پروژه**

* نظارت لحظه‌ای بر عملیات ناوگان.
* ارائه ابزارهای تحلیل و گزارش برای عملکرد ناوگان.
* طراحی رابط کاربری ماژولار و مقیاس‌پذیر برای انطباق با نیازهای جدید تجاری.

# **معماری سیستم در بخش فرانت‌اند**

سیستم از معماری برنامه تک‌صفحه‌ای (SPA) مبتنی بر React پیروی می‌کند. این معماری شامل:

* فریم‌ورک فرانت‌اند: React.js
* مدیریت وضعیت: Redux Toolkit برای وضعیت سراسری و React Context برای مدیریت وضعیت محلی
* ارتباط با API:Axios برای درخواست‌های HTTP
* ارتباط با خودرو: Socket برای حفظ ارتباط Real Time با فرستنده اطلاعات
* طراحی کامپوننت: اصول طراحی اتمیک همراه با ساختار مبتنی بر ویژگی

# **لایه‌بندی کامپوننت‌ها**

سیستم به لایه‌های مختلف برای جداسازی مسئولیت‌ها و افزایش مقیاس‌پذیری تقسیم شده است:

1. **Presentation Layer:**

* شامل کامپوننت‌های UI قابل استفاده مجدد (دکمه‌ها، جداول، کارت‌ها و غیره).

1. **Container Layer**

* کامپوننت‌هایی که مسئول دریافت داده و مدیریت وضعیت هستند.
* اتصال کامپوننت‌های نمایش به منطق کسب‌وکار.

1. **Service Layer**

* شامل توابع برای درخواست‌های API و دستکاری داده‌ها.
* جداسازی واضح بین رابط کاربری و یکپارچگی بک‌اند را تضمین می‌کند.

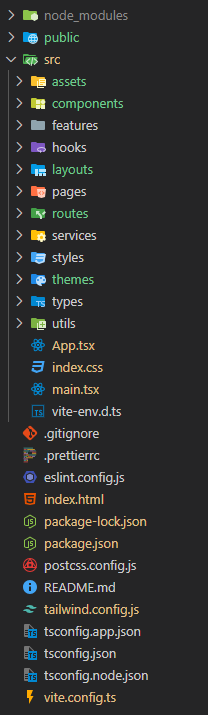
1. **State Layer**

* مدیریت وضعیت مرکزی با Redux Toolkit برای وضعیت میان‌ویژگی‌ها.
* مدیریت وضعیت محلی با useState و useReducer.

# **ساختار پوشه‌بندی مبتنی بر ویژگی**

این پروژه از ساختار ماژولار مبتنی بر ویژگی پیروی می‌کند تا فایل‌های مرتبط به‌صورت جداگانه گروه‌بندی شده و قابلیت نگهداری و مقیاس‌پذیری افزایش یابد.

## **ساختار پوشه‌ها**



## **توضیحات بیشتر**

|  |  |
| --- | --- |
| نام پوشه | فایل‌های موجود |
| Public | تمامی فایل‌های عکسی که به شکل استاتیک مورد استفاده قرار میگیرند باید در این پوشه باشند. این امر برای راحتی بیشتر در آدرس‌دهی عکس‌ها مناسب‌تر است، به این صورت که اگر یک عکس با نام image-1.png در پوشه public وجود داشته باشد، برای آدرس‌دهی آن در تگ img نیازی نیست به صورت “public/image-1.png” عمل کنیم و فقط با نوشتن “image-1.png” در اتریبیوت src به نتیجه دلخواه خواهیم رسید. |
| Assets | تمای فایل‌های مربوط به فونت‌های استفاده شده در پروژه در این قسمت قرار میگیرد. |
| Components | تمامی کامپوننت‌های که به شکل مشترک در سراسر پروژه استفاده شده و محدود به یک ویژگی، صفحه یا کامپوننت خاصی نیستند. مثال:   * Header * Footer * … |
| Features | هر feature میتواند شامل تمامی پوشه‌های نام برده شده باشد. هدف از ایجاد این پوشه، متمرکز کردن تمام فایل‌های مربوط به یک قابلیت به منظور جلوگیری از شلوغ و ناخوانا شدن فایل‌های کلی پروژه است. به این صورت که برای مثال موارد مربوط به state، service، theme، hook و contextهای مورد نیاز صفحه ورود، در پوشه‌های مربوطه و همه آنها در پوشه login قرار میگیرند. |
| Hooks | تمامی custom hookهای مشترک در سراسر پروژه در این قسمت قرار خواهند گرفت. |
| Layouts | تمامی layoutهای مشترک در سراسر پروژه در این قسمت قرار خواهند گرفت. این layoutها به منظور استفاده در routing ساخته میشوند، به این صورت که برخی routeهای خاص نیاز به این دارند که ساختار UI متمایزی نسبت به باقی صفحات داشته باشند، مانند صفحه اطلاعات کاربر که ممکن است از ساختار طراحی جامع داشبورد پیروی نکند. |
| Pages | تمامی pageهای استفاده شده در سیستم routing در این قسمت قرار خواهند گرفت. |
| Routes | تمامی routeهای استفاده شده در سیستم routing در این قسمت قرار خواهند گرفت. |
| Services | تمامی ارتباطات با سرور از طریق API که به شکل مشترک در سراسر پروژه استفاده میشوند، در این قسمت قرار خواهند گرفت. |
| Styles | تمامی styleهای مشترک در سراسر پروژه در این قسمت قرار خواهند گرفت. |
| Themes | تمامی themeهای مشترک کتابخانه Material UI در سراسر پروژه در این قسمت قرار خواهند گرفت. |
| Types | تمامی typeهای مشترک در سراسر پروژه در این قسمت قرار خواهند گرفت. |
| Utils | تمامی utility functionهای مشترک در سراسر پروژه در این قسمت قرار خواهند گرفت. |

# **کتابخانه‌ها و ابزارها**

## **کتابخانه‌های فرانت‌اند**

* React: کتابخانه اصلی برای ساخت UI.
* Redux Toolkit: مدیریت وضعیت.
* React Router: ناوبری و مسیریابی.
* Axios: برای درخواست‌های HTTP.
* React Query (اختیاری): واکشی داده و کشینگ.
* Material UI: کتابخانه کامپوننت آماده
* Date-FNS: مدیریت اعداد، تاریخ‌ها و زمان‌ها

## **استایل‌دهی**

* CSS Modules / Styled Components / SASS: روش استایل‌دهی ماژولار.
* TailwindCSS: برای CSS کاربردی.

## **تست**

* Jest و React Testing Library: برای تست واحد و یکپارچه.
* Cypress: برای تست‌ها E2E.

# **گردش‌کار توسعه**

## **فرآیند توسعه ویژگی**

* مشخص کردن نیازمندی‌ها ویژگی‌ها.
* ایجاد پوشه جدید در features.
* ساخت کامپوننت‌ها، سرویس‌ها، و اسلایس‌های مختص ویژگی.

## **کنترل نسخه**

* استفاده از Git با استراتژی branch برای هر ویژگی.
* نام‌گذاری branch به شکل feature/<feature-name>

## **تست**

* نوشتن تست‌های واحد برای کامپوننت‌ها و اسلایس‌ها.
* اطمینان از پوشش‌دهی تست بالای ۸۰٪.

# **استانداردهای کدنویسی و بهترین روش‌ها**

1. **نام‌گذاری فایل‌ها:** استفاده از PascalCase برای کامپوننت‌ها و camelCase برای فایل‌ها.
2. **ساختار کامپوننت‌ها:**

* استفاده از کامپوننت‌های فانکشنال همراه با هوک‌های React.
* پیروی از اصل مسئولیت واحد.

1. **مدیریت وضعیت:**

* استفاده از Redux فقط برای وضعیت بین ویژگی‌ها.
* اجتناب از بارگذاری بیش از حد وضعیت سراسری؛ اولویت با اسلایس‌های مختص ویژگی.

1. **مدیریت خطاها:**

* مدیریت خطاهای API در سرویس مربوطه.
* استفاده از Error Boundaries برای خطاهای سطح UI.

# **بهبودهای آینده**

پیاده‌سازی ویژگی‌های برنامه وب پیشرفته (PWA) برای عملکرد آفلاین.