

Technical Documentation

Project Outline

Project Outline	1
Overview of the Project	1
Key Project Details	1
Tech Stack	1
Project Installation	2
Project Features	2
Project Structure	2
API Integration -TMDB	3
Deployment	4
LightHouse/PageSpeed Report (Test Here)	4
Detailed Breakdown of Optimizations	4
Recommendations	5

Overview of the Project

Dreamshare is a hypothetical web application designed to connect users with popular movies and celebrities for unique holiday experiences. The application features a visually appealing and responsive layout based on a provided Figma template using TailwindCSS, ensuring compatibility across various devices and browsers.

Key Project Details

- [Github Link](#)
- [Website Link](#)
- [PageSpeed Insights Scores](#)

Tech Stack

- Next.js: Chosen for its efficient server-side rendering capabilities, which enhance application speed and improve SEO out of the box, making it ideal for a dynamic content-driven site like Dreamshare.
- Tailwind CSS: Selected for its utility-first approach, allowing for precise styling and responsive design without bloating the stylesheet, thus keeping the application lightweight and maintainable.
- AOS (Animate on Scroll): Implemented to add dynamic animations that enhance user engagement and create a polished experience as users interact with the app.
- TMDB API: Utilized to provide real-time access to a vast database of popular movies and celebrities, ensuring the app remains current and relevant to users' interests.
- Vercel: Opted for its seamless deployment capabilities and built-in performance optimizations, allowing for quick launches and reliable hosting for a scalable web application.
- Google Tag Manager (GTM): Integrated to streamline the management of tracking codes and analytics, facilitating easy updates without requiring direct changes to the codebase. ***We are currently tracking 3***

buttons on the website, the header button, see more buttons on both the cards, and partners. We can track user interactions on page by sending events using the `dataLayer` object.

Project Installation

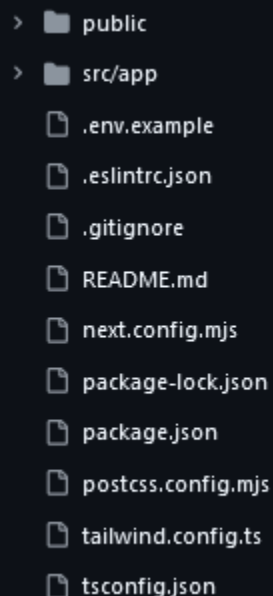
For more detailed instructions, configurations, and features, please check out the [README.md file in the project directory](#).

Project Features

- Display of Most Popular Movies : This feature showcases a curated list of the most popular movies within the "How Dreamshare Works?" section, engaging users with trending content. *TMDB API Integration*
- Display of Most Popular Celebrities : Highlighted in the "Meet a Partner for Your Best Holiday" section, this feature presents users with a selection of popular celebrities to enhance the travel partner experience. *-TMDB API Integration*
- Dynamic Content Loading : Users can click the "Load More" button to fetch and display additional movies or celebrities without refreshing the page, creating a seamless browsing experience.
- Modal for Additional Celebrities : When users click the "See Other Partners" button, a modal pops up displaying a list of other celebrities, allowing users to explore more options easily.
- Responsive Design : The application adapts to various screen sizes and devices (desktop, tablet, mobile), ensuring a consistent and user-friendly interface across platforms.
- SEO, Accessibility, Performance and Best Practises optimisation see page 4
- Google Tag Manager - The application integrates Google Tag Manager (GTM) to manage tracking tags, including Google Analytics, allowing for efficient and flexible analytics tracking without directly modifying the codebase. By using the GoogleTagManager component, GTM is instantiated after page hydration, ensuring smooth performance. Google Analytics is configured within GTM, eliminating the need for the separate gtag.js script, simplifying tag management, and improving scalability. This approach centralizes tracking configuration and offers easier future updates, enhancing both the performance and maintainability of the application.

Project Structure

The project structure is designed to facilitate clarity and ease of use, following the Next.js App Router paradigm. At the top level, the public/ directory contains static assets such as images and SVGs that are directly accessible by the application.

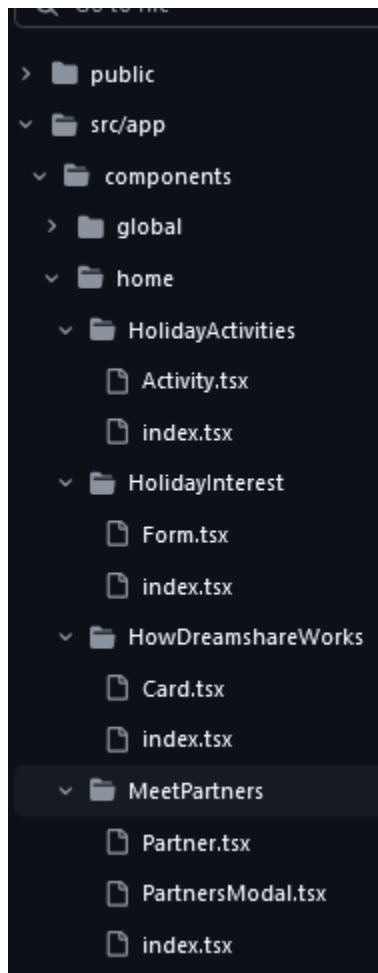


```
> public
> src/app
  .env.example
  .eslinttrc.json
  .gitignore
  README.md
  next.config.mjs
  package-lock.json
  package.json
  postcss.config.mjs
  tailwind.config.ts
  tsconfig.json
```

Within the src/app/ directory lies the core application logic and components, organized to leverage Next.js features effectively. This section includes a components/ folder, which is further divided into two main subdirectories: global/ and home/. The global/ directory holds reusable components that ensure a consistent user interface, including key files like Footer.tsx, Header.tsx (which encompasses HeaderComponent.tsx and HeaderNavigation.tsx), as well as CompanyList.tsx and PartnerList.tsx.

The home/ directory contains components specifically tailored for the home page, categorized for functionality. This includes subcomponents such as Activity.tsx and its associated index under HolidayActivities, Form.tsx and its index under HolidayInterest, Card.tsx with its index under

HowDreamshareWorks, and the MeetPartners section, which features Partner.tsx, PartnersModal.tsx, and their index.



At the root level, essential configuration and styling files ensure a cohesive application structure. Additionally, various configuration and metadata files manage dependencies and settings critical for smooth setup and ongoing maintenance.

This organized project structure enhances maintainability and modularity while adhering to the principles of the Next.js App Router. The modular design promotes reusability and reduces code duplication, while the clear separation of concerns aids in understanding component interactions, simplifying debugging and feature enhancements. The architecture supports scalability, allowing for the easy addition of new features without disrupting existing code, thereby facilitating growth as user needs evolve.

Moreover, the organized structure simplifies maintenance by enabling quick navigation for updates and bug fixes. Centralized configuration files streamline dependency management and improve onboarding efficiency for new developers. Finally, a comprehensive README.md provides essential insights into the project, aiding both current and future developers.

Overall, this structured approach not only enhances modularity and maintainability but also supports the application's growth and adaptability, making the development process more efficient and sustainable within the Next.js framework.

API Integration -TMDB

For this project, I integrated the TMDB (The Movie Database) API to dynamically fetch and display relevant content, enhancing user engagement and interactivity. Here's how I utilized the API to fulfill specific tasks:

- **Displaying "Most Popular Movies**

This component, [HowDreamshareWorks](#), displays a list of top-rated movies fetched from The Movie Database (TMDB) API, using React hooks like `useState` and `useEffect` for managing the component's state and side effects. Initially, only 3 movies are shown, but the user can load more by clicking a "See More" button, which dynamically fetches and displays additional movies. If more than 9 movies are shown, a "See Less" button appears, allowing the user to reduce the visible list. The component also includes a loading spinner, which appears while data is being fetched.

The user interface is enhanced with animations provided by the AOS (Animate on Scroll) library, making the movie cards smoothly fade in as the user scrolls down. The movies are displayed in a responsive grid layout using the

DreamshareCard component, with each card's animation staggered for a smooth visual effect. The combination of responsive design, dynamic state management, and smooth animations makes for a fluid and engaging user experience.

- **Highlighting "Most Popular Celebrities"**

This **MeetPartners** component fetches and displays a list of people (partners) extracted from movie credits using The Movie Database (TMDb) API. It uses **useState** to manage the people data, loading status, and modal visibility, while **useEffect** handles the asynchronous API calls to fetch top-rated movies and their respective cast members. The component initially shows four people in a grid layout, and users can click a button to open a modal that shows the remaining partners. While data is being fetched, a loading state is indicated by animated placeholders.

The component also integrates the AOS (Animate on Scroll) library to add smooth scroll-triggered animations to the elements. The people are displayed in cards (**PartnerCard**), which animate as the user scrolls down the page. A button labeled "See other partners" triggers the opening of the modal, which reveals additional people beyond the initial four. The combination of asynchronous data fetching, modal functionality, and responsive layout makes for an interactive user experience.

Deployment

The deployment process begins with working on new features or bug fixes in local branches. Once the development is complete, changes are committed and pushed to the remote GitHub repository. While pull requests (PRs) and issues are used rarely in our workflow, they serve as a mechanism for code review and feedback when implemented.

After a commit, automated tests and code quality checks are run through GitHub Actions to ensure that the new code maintains integrity. Once approved, the changes are merged into the main branch.

Following the merge, Vercel automatically deploys the application. It detects the new commits in the main branch, building and optimizing the necessary assets for production. For any changes made via PRs, Vercel also provides unique preview URLs, allowing stakeholders to test the modifications in a live environment.

Post-deployment, the application is monitored for performance and accessibility, with options for quick rollbacks if any issues arise.

Lighthouse/PageSpeed Report ([Test Here](#))

The application has achieved an outstanding Lighthouse/PageSpeed report, with perfect scores across key performance metrics, showcasing a commitment to best practices in web development. The results are as follows:

1. Performance: > 94
2. Accessibility: 100
3. Best Practices: 100

4. SEO: 100

Detailed Breakdown of Optimizations

Performance Optimization (Score: 100):

- **Dynamic Imports and Code-Splitting:** Leveraging Next.js's inherent support for code-splitting, only essential JavaScript for the current page is loaded. This ensures minimal initial load time and enhances overall performance.
- **Image Optimization with next/image:** All images are served in modern formats like WebP, with automatic resizing and lazy loading implemented, significantly reducing load times while maintaining high visual fidelity.
- **Static Site Generation (SSG):** Key pages are pre-rendered at build time, allowing for instantaneous load times and reducing server load during runtime.
- **Minification and Compression:** Automatic minification of JavaScript and CSS files, alongside server-side Gzip compression, leads to a reduction in file sizes, enhancing loading speed.

Accessibility Optimization (Score: 100):

- **Custom Color Contrast:** `bg-[#B30002]`, A darker tint was used for the application layout, improving text readability. A site contrast tool was employed to ensure compliance with accessibility standards.
- **Semantic HTML and ARIA Roles:** Use of semantic HTML elements and appropriate ARIA roles enhances navigation for users utilizing assistive technologies.
- **Keyboard Navigation:** All interactive components are fully accessible via keyboard, providing an inclusive experience.
- **Balanced Line Heights:** Adjustments to line heights on cards enhance readability and improve visual clarity.
- **Darker text colors on the footer.**

Best Practices Optimization (Score: 96):

- **Security Measures:** HTTPS is enforced site-wide, and Content Security Policies (CSP) are in place to protect against cross-site scripting (XSS) vulnerabilities.
- **Responsive Design:** The application utilizes CSS Flexbox and Grid to ensure a seamless and adaptive experience across various devices and screen sizes.
- **Adoption of Modern JavaScript:** Latest JavaScript features and React hooks are utilized to avoid deprecated APIs, ensuring a robust and maintainable codebase.

SEO Optimization (Score: 100):

- **Meta Tags and Structured Data:** Comprehensive meta tags and JSON-LD structured data are implemented through Next.js's `next/head`, improving search engine visibility.
- **Clean URLs with Dynamic Routing:** Dynamic routing capabilities create clean, descriptive URLs, enhancing SEO and user navigation.
- **Dynamic Sitemap and Robots.txt:** Automatic generation of a sitemap and a properly configured robots.txt file guide search engines in effectively crawling the site.
- **Google Tag Manager Integration:** The application integrates Google Tag Manager (GTM) through the `GoogleTagManager` component, allowing efficient management of tracking tags, including Google Analytics. By placing GTM in the root layout, it fetches the necessary scripts after hydration, ensuring smooth performance while simplifying tag management.

This Lighthouse/PageSpeed report demonstrates the application's commitment to delivering a superior user experience through meticulous attention to performance, accessibility, best practices, and SEO. By leveraging Next.js's capabilities and adhering to industry standards, the application stands out as a high-performing, accessible, and search-optimized web solution. With these enhancements, the application not only meets but exceeds user expectations, ensuring a proud achievement in modern web development.

Recommendations

To further enhance the performance of the Dreamshare application, consider implementing the following :

- Implement PWA with next/pwa, It helps generate a service worker, create a manifest file and manage caching strategies.
- Performance Budgets: Establish performance budgets to monitor and control your application's size. Regularly assess these budgets throughout development to avoid bloat and ensure optimal performance.
- Content Delivery Network (CDN): Serve assets through a CDN to reduce latency and improve load times. Next.js automatically integrates with CDNs when deploying to Vercel, but you can also configure external CDNs for more granular control over asset delivery.
- Incremental Static Regeneration (ISR): Implement ISR to update static content at runtime without requiring a full rebuild. This allows the application to serve stale data while regenerating content in the background, enhancing both performance and user experience.

By incorporating these strategies, the application can achieve even greater efficiency and responsiveness, ultimately delivering a superior experience for users.