Project Documentation

This Next.js app, created with `create-next-app`, is launched via `npm/yarn/pnpm/bun run dev` and accessible at `http://localhost:3000`. Edit `app/page.tsx` for changes. It uses `next/font` for optimized font loading. See the Next.js docs (https://nextjs.org/docs) and deployment guide (https://nextjs.org/docs/deployment) for more information. Deploy easily with Vercel (https://vercel.com/new).

AnimatedText Component

Description: A React component that displays text word-by-word with a customizable animation delay.

Props:

* `text` (string, required): The text to animate.
* `delay` (number, optional): The delay (in milliseconds) between displaying each word. Defaults to 50ms.

Functionality:

The component splits the input text into words and displays them sequentially using `setInterval`. A cleanup function clears the interval on unmount. The `removeLastWord` function removes the last word, likely for stylistic purposes (trailing space removal).

Usage:

```jsx
import AnimatedText from './AnimatedText';

<AnimatedText text="This is animated text" delay={100} />
```

Module: `AnimatedText.module.css` (styles not detailed)

FAQs Component Documentation

This React component displays a frequently asked questions (FAQ) section. It uses an accordion component (`@/components/ui/accordion`) to show/hide answers. Data is fetched from `../data/data.js` (presumably an array of objects, each with `trigger` and `content` properties). Styling is applied using Tailwind CSS. The component is responsive, adjusting layout for different screen sizes.

Footer Component Documentation

This React component renders a website footer. It uses Tailwind CSS for styling.

Features:

* **Branding:** Displays "InkQuill" logo.
* **Navigation:** Renders links from `../data/data.js` (presumably containing `{title, link}

` objects).

* **Legal:** Provides "Terms" and "Policies" links.

* **Contact:** Includes a "Contact Us" button (from `../ui/Button.js`).

Props: None.

Dependencies:

* `react`

* `next/link`

* `../data/data.js` (for navigation links)

* `../ui/Button.js` (for the contact button)

Styling: Uses Tailwind CSS classes for responsive layout and styling. The footer has rounded corners and a blue background.

Hero Component Documentation

This React component renders a hero section for a website.

Features:

*   Responsive design using Tailwind CSS.

*   Large headline with highlighted text.

*   Supporting paragraph.

*   Button linking to `/summarizer`.

*   Hero image displayed on larger screens.

Props: None.

Dependencies:

*   `react`
*   `next/image`
*   `next/link`
*   `../ui/Button` (custom button component)
*   `../ui/hero-image/hero-image.svg` (hero image)

Usage:

```jsx
import Hero from './Hero';

// ... in your page component
<Hero />
```

This Next.js component `NavBar` renders a responsive navigation bar. It uses a hamburger menu for mobile responsiveness. The sidebar closes when clicking outside it. It utilizes custom UI components (`Button`, `Hamburger`, `Cancel`) and data from `data/data.js`. The styling is done with Tailwind CSS.

Newsletter Component Documentation

This React component renders a newsletter signup form.

Features:

* Headline: "Join Our Community" with a call to action to subscribe.
* Image: A newsletter-themed image (on larger screens).
* Form: Contains input fields for name and email, and a submit button. Uses a custom `Input` and `Button` component.
* Responsive Design: Adapts layout for different screen sizes.

Props: None.

Dependencies:

* `react`
* `next/image`
* `@/components/ui/input` (Custom Input Component)
* `../ui/Button` (Custom Button Component)

Styling: Uses Tailwind CSS for styling. Assumes an `azure-blue` color is defined.

Usage:

```jsx
import Newsletter from './Newsletter';
```

// ... in your component ...
<Newsletter />
```

## Paraphrase Component Documentation

This React component (`Paraphrase.js`) displays a section promoting paraphrasing. It features:

* **Headline:** "Paraphrase Notes"
* **Descriptive Text:** Explains paraphrasing and its benefits.
* **Call to Action:** A button linking to a summarizer tool ("/summarizer").
* **Image:** An illustrative image (`/images/summarize.png`).

Uses Tailwind CSS for styling and Next.js for routing and image import. The layout is responsive, adjusting for different screen sizes.

## Summarize Component Documentation

This React component displays a marketing section for a note summarization feature ("InkQuill"). It features:

* **Layout:** A flexbox section with an image and text content, responsive for different screen sizes.
* **Image:** Shows a relevant image (`/images/summarize.png`).
* **Text:** Highlights the feature's AI-powered summarization capabilities and benefits for efficient learning.
```

* **Call to Action:** A button linking to the `/summarizer` page.
* **Styling:** Uses Tailwind CSS classes for styling and responsiveness. Employs custom components (`Button` from `../ui/Button`).

This file `index.js` exports React components for a webpage: `NavBar`, `Hero`, `Summarize`, `Paraphrase`, `FAQs`, `Newsletter`, and `Footer`. Each component likely represents a section of the page.

ContactUs Component Documentation

This React component renders a contact form.

Features:

* **Navigation Bar:** Includes a NavBar component at the top.
* **Hero Section:** A section with a heading "Contact Us" and a brief description.
* **Form & Image:** A responsive layout displaying a contact form alongside an image (on larger screens).
* **Form Submission:** Uses `formsubmit.co` for form submissions. Includes name, email, and message fields. All fields are required.

Dependencies:

* `react`
* `next/image`
* Custom components: `NavBar`, `Input`, `Button` (paths should be adjusted according to your project structure).

Styling: Uses Tailwind CSS for styling. Color variables (`azure-blue`, `cotton-white`) are assumed to be defined elsewhere.

File Structure (Assumed):

```
components/
  ui/
    Button.js
    Input.js
components/
  NavBar.js
pages/
  contact.js (this file)
public/
  images/
    contact.svg
```

Documentation: Navigation & FAQs Data

This module exports two constants:

`links`: An array of navigation links. Each object has `title` (string) and `link` (string, relative path).

`faqs`: An array of frequently asked questions. Each object has `trigger` (string, the question) and `content` (string, the answer).

This Next.js `app` directory root layout defines a basic HTML structure. It uses the `Inter` font from Google Fonts and includes a custom CSS file (`globals.css`). The `<body>` includes the application content (`children`) and applies a `kanit-regular` class (presumably defined in `globals.css`). Metadata includes a title and description.

Loading Component Documentation

This React component displays a simple "Loading..." message centered on the page.

Component: `Loading`

Import: `import Loading from './Loading';`

Usage: `<Loading />`

Props: None

Output: A `<div>` element with the class `text-center` containing the text "Loading...".

This Next.js app renders a webpage with sections for: navigation (`NavBar`), a hero section (`Hero`), an about section describing "InkQuill" (a note-taking assistant), note summarization (`Summarize`), paraphrasing (`Paraphrase`), FAQs (`FAQs`), a newsletter signup (`Newsletter`), and a footer (`Footer`). Styling uses Tailwind CSS. The `useRouter` hook is imported but not used. The `client` directive indicates the

component is rendered on the client-side.

ParaphraseApi Component Documentation

This React component `ParaphraseApi` uses a paraphrasing API (endpoint defined in `process.env.PARAPHRASE_API`).

Functionality:

1. **Retrieves text:** Gets input text from a global store (`useClientStore`).
2. **Sends POST request:** Sends the text to the paraphrasing API.
3. **Handles response:** Updates component state with the paraphrased text (`paraphrasedText`) upon successful response. Handles errors gracefully.
4. **Manages loading state:** Uses `paraphraseLoading` to indicate API request status.

Exports:

* `paraphraseText`: Function to trigger the paraphrasing API call.
* `paraphrasedText`: State variable holding the paraphrased text.
* `setParaphrasedText`: Function to update `paraphrasedText` (though directly manipulating this outside the component is discouraged).
* `paraphraseLoading`: Boolean indicating API request progress.

Dependencies:

* `axios`: For making HTTP requests.
* `useClientStore`: Custom hook for accessing application state (likely a Pinia or

Zustand store).
*   `useState`: React Hook for managing component state.

Configuration:

The API endpoint is configured via the environment variable `PARAPHRASE_API`. Failure to set this will result in an empty string being used, causing an error.

This Next.js component `Paraphraser` provides a UI for paraphrasing text. It uses a custom hook `useClientStore` for state management, `ParaphraseApi` for API calls (implementation not shown), and `AnimatedText` for animated text display. The user pastes text, clicks "Paraphrase," and the result is displayed, with text-to-speech functionality. Navigation links to a summarizer are included. Styling is done with Tailwind CSS.

Summarizer Page Documentation

This React component renders a page with navigation (`NavBar`), links to summarization and paraphrasing tools, and options to upload a file or paste text for summarization. Styling uses Tailwind CSS. The `use client` directive indicates a client-component.

SummarizeApi Component Documentation

This React component `SummarizeApi` handles summarizing text using an external API.

Functionality:

1. **Retrieves text:** Gets text data from a global store (`useClientStore`).
2. **Input validation:** Checks if the text length is sufficient (at least 250 characters). Displays an error if not.
3. **API call:** Sends a POST request to the `SUMMARIZE_API` endpoint (environment variable). The request body contains the text to be summarized.
4. **Response handling:** Updates the component's state with the summarized text from the API response or handles errors.
5. **State management:** Uses `useState` for managing loading state (`summarizeLoading`), error state (`errorLength`), and summarized text (`summarizedText`).

Exports:

The component exports the following:

* `summarizeText`: Function to trigger the summarization process.
* `summarizedText`: The summarized text (string).
* `setSummarizedText`: Function to directly update the `summarizedText` state.
* `summarizeLoading`: Boolean indicating whether the summarization is in progress.
* `errorLength`: Boolean indicating whether the input text is too short.

Dependencies:

* `axios`: For making HTTP requests.
* `useClientStore`: A custom hook for accessing a global store (presumably containing

the input text).
* `useState`: React hook for managing state.

Environment Variable:

Requires the `SUMMARIZE_API` environment variable to be set with the API endpoint URL.

This Next.js component (`PasteText`) provides a text summarization interface. Users paste text into a textarea. Clicking "Summarize" sends the text to a summarization API (`SummarizeApi`). The summarized text is displayed, with an option to play it using text-to-speech. Error handling is included for text shorter than 250 words. Navigation links to other summarization/paraphrasing tools are also present. The UI uses custom components (`NavBar`, `AnimatedText`, `Speaker`) and styling. Client-side state management is handled by `useClientStore`.

UseOcrApi Hook Documentation

This hook (`UseOcrApi`) handles OCR using an external API.

Functionality:

1. **Receives a PDF URL:** Takes a PDF URL via `setFileUrl`.
2. **Sends OCR request:** Makes a POST request to the OCR API (URL from `process.env.OCR_API`). Includes API key (`process.env.OCR_API_KEY`), language, and other parameters.
3. **Parses response:** Parses the JSON response, extracts the text from

`ParsedResults[0].ParsedText`, and stores it in the global store via `setData`.

4. **Navigates:** Navigates to `/summarizer/paste-text` after successful OCR.
5. **Handles loading:** Uses `isLoading` state to manage loading status.

Exports:

* `getText`: Asynchronous function to initiate the OCR process.
* `setFileUrl`: Function to set the PDF URL.
* `fileUrl`: Current PDF URL (state).
* `isLoading`: Boolean indicating loading status.

Dependencies:

* `useClientStore` (for global state management).
* `useState` (for local state management).
* `useRouter` (for navigation).

Environment Variables:

* `OCR_API_KEY`: API key for the OCR service.
* `OCR_API`: Base URL of the OCR API endpoint.

Error Handling:

Basic error logging to the console. No specific user-facing error handling is implemented.

This React component (`Upload.js`) allows users to upload a PDF file via a Cloudinary API. The uploaded file's URL is stored in state, enabling subsequent OCR processing (via the `UseOcrApi` custom hook). The UI includes a file input, upload button (disabling during upload), and links to summarizer and paraphraser pages. Cloudinary credentials are sourced from environment variables.

Button Component

Description: A reusable React button component with customizable background color.

Props:

* `label` (string, required): The text displayed on the button.
* `whiteBg` (boolean, optional): If `true`, button has a white background; otherwise, it uses a blue background (with styling variations on medium screens and hover).

Example Usage:

```jsx
<Button label="Click Me" whiteBg={true} />
<Button label="Submit" />
```

Cancel Component Documentation

This React component renders a simple 'X' shaped cancel icon.

Import:

```javascript
import Cancel from './Cancel';
```

Props: None. The icon is hardcoded to be 30x30 pixels with a blue (#007fff) stroke.

Usage:

```javascript
<Cancel />
```

This will render a 30x30 pixel blue 'X' icon.

Speaker Component Documentation

This React component renders a speaker icon as an SVG.

Component: `Speaker`

Import: `import Speaker from './Speaker';`

Props: None

Functionality: Displays a stylized speaker icon, 24x24 pixels, white fill, using SVG paths. The icon consists of a main speaker body and lines suggesting sound waves.

Example Usage: `<Speaker />`

This code provides a custom React component for an accordion using the `@radix-ui/react-accordion` library and Lucide icons. It exports four components:

* **`Accordion`**: The root accordion container.
* **`AccordionItem`**: A single accordion item. Adds a bottom border.
* **`AccordionTrigger`**: The clickable header of an item. Includes a chevron icon that rotates on open/close. Uses Tailwind CSS for styling.
* **`AccordionContent`**: The collapsible content area of an item. Uses Tailwind CSS for styling and animations.

The components use React's `forwardRef` for better accessibility and composition. They leverage `cn` (likely a utility function for className concatenation) and Tailwind CSS for styling. The animation classes (`animate-accordion-up`, `animate-accordion-down`) are assumed to be defined elsewhere (likely in a CSS file).

Next.js Configuration

This Next.js configuration file (`next.config.js`) defines:

* **`images.domains`**: Allows image imports from `imgs.search.brave.com`.
* **`env`**: Sets environment variables accessible throughout the application:
    * `OCR_API_KEY`: API key for OCR service.

* `CLOUDINARY_PRESET_KEY`, `CLOUDINARY_CLOUD_NAME`: Cloudinary credentials.
* `PARAPHRASE_API`, `SUMMARIZE_API`: API endpoints for paraphrasing and summarization services (Inkquill).
* `OCR_API`: API endpoint for OCR service (OCR.space).

The config is exported as the default export.

Inkquill Documentation (v0.1.0)

Description: A private Next.js application (likely a web app).

Dependencies: React, Next.js, Tailwind CSS, Zustand (state management), Lucide (icons), Framer Motion (animations), Axios (HTTP requests), Cohere AI (likely for AI features), OCR APIs (optical character recognition), and various utility libraries.

Scripts:
* `dev`: Starts the development server.
* `build`: Builds the production application.
* `start`: Starts the production server.
* `lint`: Runs the linter.

Note: Private repository, limited information available. Further details require access to the repository's codebase.

This code exports a PostCSS configuration object. It uses `tailwindcss` plugin without

any additional options. The `@type` annotation ensures type safety within a TypeScript project.

Zustand Store with Local Storage Persistence

This code provides a Zustand store (`useStore`) managing a string (`data`) and a setter function (`setData`). `setData` updates both the store and local storage. `useClientStore` is a custom hook that loads data from local storage on component mount. It uses a ref to prevent infinite loops. Requires `zustand` and `react`.

This Tailwind CSS configuration file uses dark mode based on a class, scans specified directories for content, centers containers with 2rem padding and a 1400px width on 2xl screens. It defines custom colors using CSS variables, rounded corners based on a `--radius` variable, and custom keyframes/animations for accordions. The `tailwindcss-animate` plugin is included.

This `tsconfig.json` configures a TypeScript project for Next.js. It uses ES modules (`module: "esnext"`), supports JS files (`allowJs: true`), enables strict type checking (`strict: true`), and utilizes Next.js plugins. The `paths` alias maps `@/*` to the project's root directory for convenient imports. `noEmit: true` prevents TypeScript from generating `.js` files; incremental compilation is enabled. It includes TypeScript and JSX files, excluding `node_modules`.

API Documentation

This Node.js app uses the Cohere API for text summarization and paraphrasing. It's an Express.js server with CORS enabled.

Endpoints:

* `/api/summarize` (POST): Summarizes input text using Cohere's `summarize()` method. Requires a JSON body with a `text` field. Returns a JSON object with `success` (boolean) and the Cohere summarization response.

* `/api/paraphrase` (POST): Paraphrases input text using Cohere's `generate()` method. Requires a JSON body with a `text` field. Returns a JSON object with `success` (boolean) and the Cohere generation response.

Error Handling: A middleware function handles errors, returning JSON with `success: false`, `statusCode`, and `message`.

Environment: Requires a `COHERE_API_KEY` environment variable.

Dependencies: `cohere-ai`, `express`, `cors`, `dotenv`.

Server: Listens on port 8000.

Server Documentation

Name: server

Version: 1.0.0

Description: (None provided)

Main file: index.js

Dependencies: express, cors, dotenv, nodemon, cohere-ai

Scripts:

* `dev`: Starts the server using nodemon for development.

To run:

1. `npm install`
2. `npm run dev`

This is a basic Node.js server using Express. Further documentation is needed within the `index.js` file itself for detailed functionality.