= /Users/dines/Documents/GitHub/stackunderflow-ui/src/App.css =

```
/ src/App.css /
.app-container {
display: flex;
flex-direction: column;
min-height: 100vh;
}

.main-content {
max-width: 1100px; / Set a max width for the content area /
width: 100%;
margin: 0 auto; / Center the content horizontally /
padding: 1rem 1.5rem; / Add some padding on the sides and top/bottom /
flex-grow: 1;
}

/ --- Navbar Styles --- /
.navbar {
background-color: #ffffff;
border-bottom: 1px solid #d6d9dc;
padding: 0 1.5rem; / Match side padding of main-content /
display: flex;
align-items: center;
justify-content: space-between; / Pushes items to the ends /
height: 50px;
}

.navbar-left, .navbar-right {
display: flex;
align-items: center;
gap: 1rem; / Adds space between items /
}

.navbar a {
color: #525960;
text-decoration: none;
font-size: 14px;
padding: 6px 12px;
border-radius: 20px;
```

```
transition: background-color 0.2s;
}

.navbar a:hover {
background-color: #e3e6e8;
}

.navbar .so-logo {
font-weight: bold;
color: #000000;
}

.navbar .so-logo:hover {
background-color: transparent; / Don't highlight the logo on hover /
}

/ Style the button to match Stack Overflow /
.navbar button, .button-primary {
background-color: #0A95FF;
color: white;
border: 1px solid transparent;
border-radius: 5px;
padding: 8px 12px;
font-size: 13px;
cursor: pointer;
transition: background-color 0.2s;
}

.navbar button:hover, .button-primary:hover {
background-color: #0077CC;
}

/ --- Form Styles --- /
.form-container {
background-color: #ffffff;
padding: 24px;
border: 1px solid #d6d9dc;
border-radius: 5px;
max-width: 500px;
margin: 2rem auto;
}
```

```
.form-container h2 {
font-size: 21px;
font-weight: 400;
margin-top: 0;
margin-bottom: 24px;
}

.form-group {
margin-bottom: 1rem;
}

.form-group label {
display: block;
font-weight: 600;
margin-bottom: 5px;
}

.form-group input, .form-group textarea {
width: 100%;
padding: 8px 10px;
border: 1px solid #babfc4;
border-radius: 5px;
font-size: 14px;
}

.form-group input:focus, .form-group textarea:focus {
outline: none;
border-color: #6cbbf7;
box-shadow: 0 0 0 4px rgba(0, 149, 255, 0.15);
}

.form-error {
color: #d0393e;
font-size: 13px;
margin-top: 5px;
}

/ --- Question List Styles --- /
.question-card {
background-color: #ffffff;
padding: 16px;
border: 1px solid #d6d9dc;
border-radius: 5px;
```

```
margin-bottom: 1rem;
}

.question-card h3 {
margin: 0 0 8px 0;
font-size: 18px;
font-weight: 400;
}

.question-card h3 a {
text-decoration: none;
color: #0077CC;
}

.question-card h3 a:hover {
color: #0A95FF;
}

.question-card p {
margin: 0;
font-size: 13px;
color: #6a737c;
}

/ --- Question Detail Styles --- /
.question-detail-body {
font-size: 15px;
line-height: 1.5;
}

.answer-card {
padding: 16px;
border-top: 1px solid #d6d9dc;
margin-top: 1rem;
}

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/index.js =
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { AuthProvider } from './context/AuthContext';
const root = ReactDOM.createRoot(document.getElementById('root'));
```

```
root.render(
<React.StrictMode>
{/ Wrap the App component with the AuthProvider /}



</React.StrictMode>
);

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/index.css =
/ src/index.css /
body {
margin: 0;
font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
sans-serif;
-webkit-font-smoothing: antialiased;
-moz-osx-font-smoothing: grayscale;
background-color: #f1f2f3 ; / A light grey background, similar to Stack Overflow /
color: #242729;
}

/ A universal reset for better layout control /
, ::before, *::after {
box-sizing: border-box;
}

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/App.js =
import React from 'react';
import { BrowserRouter as Router, Routes, Route, Link } from 'react-router-dom';
import HomePage from './pages/HomePage';
import LoginPage from './pages/LoginPage';
import './App.css';
import { useAuth } from './context/AuthContext'; // Import the hook
import { useNavigate } from 'react-router-dom';
import ProtectedRoute from './components/ProtectedRoute';
import AskQuestionPage from './pages/AskQuestionPage';
import QuestionDetailPage from './pages/QuestionDetailPage';
import RegisterPage from './pages/RegisterPage';
import ProjectDetailsPage from './pages/ProjectDetailsPage';
// Create a new component for the navigation logic
const Navigation = () => {
```

const { isLoggedIn, logout } = useAuth();
const navigate = useNavigate();

```
const handleLogout = () => {
    logout(); // Clear the auth state
    navigate('/login'); // Redirect to login page
};

return (
    <header className="navbar">
  <div className="navbar-left">
    <Link to="/" className="so-logo">Stack Underflow</Link>
    <Link to="/">Questions</Link>
    <Link to="/about">About</Link>
  </div>
  <div className="navbar-right">
    {isLoggedIn ? (
      <>
        <Link to="/ask">Ask Question</Link>
        <button onClick={handleLogout}>Logout</button>
      </>
    ) : (
      <>
        <Link to="/login">Login</Link>
        <Link to="/register" className="button-primary" style={{color:
'white'}}>Sign up</Link>
      </>
    )}
  </div>
</header>
);
```

}

function App() {

return (

```
      <div className="main-content">
      <Routes>
        <Route path="/" element={<HomePage />} />
        <Route path="/login" element={<LoginPage />} />
```

```
          <Route path="/register" element={<RegisterPage />} />
          <Route path="/about" element={<ProjectDetailsPage />} />
          {/* 4. Add the new protected route */}
          <Route
            path="/ask"
            element={
              <ProtectedRoute>
                <AskQuestionPage />
              </ProtectedRoute>
            }
          />
          <Route path="/questions/:id" element={<QuestionDetailPage />} />
        </Routes>
      </div>
    </div>
  </Router>
```

);
}

export default App;

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/context/AuthContext.js =
import React, { createContext, useState, useContext, useEffect } from 'react';

// 1. Create the context
const AuthContext = createContext(null);

// 2. Create the provider component
export const AuthProvider = ({ children }) => {
const [token, setToken] = useState(localStorage.getItem('token'));

// This effect runs whenever the token state changes
useEffect(() => {
if (token) {
// If there's a token, set it in localStorage
localStorage.setItem('token', token);
} else {
// If the token is null/undefined, remove it from localStorage
localStorage.removeItem('token');
}
}, [token]);

// The login function updates the token state
const login = (newToken) => {

```
setToken(newToken);
};

// The logout function clears the token state
const logout = () => {
setToken(null);
};

// The value provided to consuming components
const value = {
token,
isLoggedIn: !!token, // A handy boolean flag
login,
logout,
};

return <AuthContext.Provider value={value}>{children}</AuthContext.Provider>;
};

// 3. Create a custom hook for easy consumption of the context
export const useAuth = () => {
return useContext(AuthContext);
};
```

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/components/AnswerForm.js =

```
import React, { useState } from 'react';
import api from '../services/api';

const AnswerForm = ({ questionId, onAnswerPosted }) => {
const [body, setBody] = useState('');
const [error, setError] = useState('');
const [success, setSuccess] = useState(''); // <-- 1. Add state for success message
const [isSubmitting, setIsSubmitting] = useState(false); // <-- 2. Add state for submission status

const handleSubmit = async (e) => {
e.preventDefault();
setError('');
setSuccess('');
setIsSubmitting(true); // <-- 3. Set submitting to true
if (!body) {
setError('Answer body cannot be empty.');
return;
}
```

```
  try {
    // The JWT is added automatically by the interceptor
    await api.post(`/questions/${questionId}/answers`, { body });
    setBody(''); // Clear the textarea
    setSuccess('Your answer has been posted!');
    // Use a timeout to give the user time to read the message before
  redirecting
      setTimeout(() => {
        onAnswerPosted(); // Notify the parent component to refetch data
      }, 1500); // 1.5 second delay

  } catch (err) {
    console.error('Failed to post answer:', err);
    setError('Failed to post answer. Please try again.');
    setIsSubmitting(false); // <-- 5. Re-enable the form on failure
  }
```

};

return (

```
  <form onSubmit={handleSubmit} style={{ marginTop: '20px' }}>
    <h4>Your Answer</h4>
    <textarea
      rows="7"
      style={{ width: '80%' }}
      value={body}
      onChange={(e) => setBody(e.target.value)}
    ></textarea>
    {error && <p style={{ color: 'red' }}>{error}</p>}
     {success && <p style={{ color: 'green' }}>{success}</p>}
    <br />
    <button type="submit" className="button-primary" disabled={isSubmitting}>
       {isSubmitting ? 'Posting Answer...' : 'Post Your Answer'}
      </button>
  </form>
    </div>
```

);
};

export default AnswerForm;

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/components/Vote.css =

/ *src/components/Vote.css* /

```
.vote-container {
display: flex;
flex-direction: column;
align-items: center;
gap: 8px; / Space between buttons and count /
margin-right: 20px; / Space between vote component and post body /
}

.vote-button {
background-color: transparent;
border: 1px solid #babfc4;
border-radius: 50%; / Makes it circular /
cursor: pointer;
width: 36px;
height: 36px;
display: flex;
align-items: center;
justify-content: center;
transition: background-color 0.2s;
}

.vote-button:hover {
background-color: #e3e6e8;
}

.vote-button.voted {
background-color: #f48024; / Stack Overflow's orange /
border-color: #f48024;
}

.vote-button.voted svg {
fill: #ffffff;
}

.vote-count {
font-size: 20px;
font-weight: 600;
color: #242729;
}

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/components/ProtectedRoute.js =
import React from 'react';
```

```
import { Navigate } from 'react-router-dom';
import { useAuth } from '../context/AuthContext';

const ProtectedRoute = ({ children }) => {
const { isLoggedIn } = useAuth();

if (!isLoggedIn) {
// If user is not logged in, redirect them to the /login page
return ;
}

// If user is logged in, render the child component (the protected page)
return children;
};

export default ProtectedRoute;

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/components/Vote.js =
import React, { useState } from "react";
import { voteOnQuestion, voteOnAnswer } from "../services/api";
import "./Vote.css"; // Import our new CSS
import { useAuth } from "../context/AuthContext";

// Simple SVG components for the arrows
const UpArrow = () => (




);

const DownArrow = () => (




);

const Vote = ({ post, postType, onVoteSuccess }) => {
const { isLoggedIn } = useAuth();

// 'userVote' can be 'UPVOTE', 'DOWNVOTE', or null
const [userVote, setUserVote] = useState(null); // NOTE: For a complete solution, you'd need
the API to tell you the user's current vote on load. We are simplifying for now.
const [error, setError] = useState("");
```

const handleVote = async (voteType) => {
if (!isLoggedIn) {
setError('You must be logged in to vote.');
return;
}
setError('');

```
    const voteApi = postType === 'question' ? voteOnQuestion : voteOnAnswer;

    try {
        // 1. Make the API call to cast the vote.
        await voteApi(post.id, voteType);

        // 2. If successful, update the local button highlight state.
        setUserVote(prevVote => (prevVote === voteType ? null : voteType));

        // 3. Call the callback function passed from the parent to trigger a
 data re-fetch.
        onVoteSuccess();

    } catch (err) {
        console.error('Failed to vote:', err);
        setError('Vote failed. Please try again.');
    }
};
```

return (

<button
className={`vote-button ${userVote === 'UPVOTE' ? 'voted' : ''}`}
onClick={() => handleVote('UPVOTE')}
title="Upvote"
>

```
        {/* Display the voteCount directly from the prop */}
        <span className="vote-count">{post.voteCount}</span>

        <button
            className={`vote-button ${userVote === 'DOWNVOTE' ? 'voted' :
 ''}`}
            onClick={() => handleVote('DOWNVOTE')}
            title="Downvote"
```

```
        >
            <DownArrow />
        </button>
        {error && <p style={{ color: 'red', fontSize: '12px', textAlign:
'center' }}>{error}</p>}
    </div>
```

);
};

export default Vote;

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/pages/ProjectDetailsPage.js =
import React from 'react';

// A simple reusable style for section headings
const sectionTitleStyle = {
borderBottom: '2px solid #0A95FF',
paddingBottom: '5px',
marginBottom: '1rem',
marginTop: '2rem'
};

const ProjectDetailsPage = () => {
return (

<h1 style={{ textAlign: 'center' }}>Project: Stack Underflow
<p style={{ textAlign: 'center', fontSize: '1.1rem', color: '#525960' }}>
A Full-Stack Stack Overflow Clone

```
  <div style={{ textAlign: 'center', margin: '1rem 0' }}>
    {/* <strong>Live Backend API:</strong>{' '}
    <a href="https://stackoverflow-api.onrender.com/api/questions"
target="_blank" rel="noopener noreferrer">
      https://stackoverflow-api.onrender.com/api/questions
    </a> */}
    {/* Add your frontend link here once it's deployed */}
    {/* <br/><strong>Live Frontend:</strong> <a href="#">Your Frontend
Link</a> */}
  </div>

  <h2 style={sectionTitleStyle}>Project Overview</h2>
  <p>
```

```
    Stack Underflow is a full-stack web application designed as a
simplified, modern clone of the popular developer Q&A platform, Stack
Overflow. This project demonstrates a complete development lifecycle, from
backend API design and implementation to a dynamic, responsive frontend user
interface. It serves as a practical showcase of key skills in modern Java
(Spring Boot) and JavaScript (React) development, including REST API design,
security, database management, and frontend state management.
  </p>

  <h2 style={sectionTitleStyle}>Core Functionalities</h2>
  <ul>
    <li><strong>User Management & Security:</strong> Secure user
registration, stateless JWT authentication, and protected routes.</li>
    <li><strong>Q&A System:</strong> Users can post, view, and answer
questions.</li>
    <li><strong>Dynamic UI:</strong> A responsive single-page application
that reacts to user authentication state.</li>
  </ul>

  <h2 style={sectionTitleStyle}>Technical Stack & Architecture</h2>
  <div style={{ display: 'flex', gap: '2rem', flexWrap: 'wrap' }}>
    <div style={{ flex: 1, minWidth: '300px' }}>
      <h4>Backend (Java / Spring Boot)</h4>
      <ul>
        <li><strong>Framework:</strong> Spring Boot 3</li>
        <li><strong>Security:</strong> Spring Security 6 with JWT</li>
        <li><strong>Database:</strong> PostgreSQL with Spring Data JPA</li>
        <li><strong>Architecture:</strong> Layered (Controller, Service,
Repository)</li>
      </ul>
    </div>
    <div style={{ flex: 1, minWidth: '300px' }}>
      <h4>Frontend (JavaScript / React)</h4>
      <ul>
        <li><strong>Framework:</strong> React 18</li>
        <li><strong>State Management:</strong> React Context API</li>
        <li><strong>Routing:</strong> react-router-dom</li>
        <li><strong>API Communication:</strong> axios</li>
      </ul>
    </div>
  </div>

  <h2 style={sectionTitleStyle}>Deployment & Hosting</h2>
  <ul>
   <li><strong>Backend API:</strong> Containerized with
<strong>Docker</strong> and deployed on <strong>Render</strong>.</li>
```

```
    <li><strong>Database:</strong> Managed <strong>PostgreSQL</strong>
  instance on <strong>Render</strong>.</li>
    <li><strong>Frontend:</strong> Deployed as a static site on
  <strong>Vercel / Netlify</strong> (once completed).</li>
  </ul>

  <h2 style={sectionTitleStyle}>Future Updates & Enhancements</h2>
  <ul>
    <li><strong>Voting System:</strong> Implementing upvote/downvote
  functionality.</li>
    <li><strong>Commenting:</strong> Adding the ability to leave comments on
  posts.</li>
    <li><strong>Search & Tagging:</strong> Implementing search and a tagging
  system for questions.</li>
    <li><strong>User Profiles:</strong> Creating pages for users to view
  their activity.</li>
  </ul>
</div>
```

);
};

export default ProjectDetailsPage;

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/pages/LoginPage.js =

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import api from '../services/api';
import { useAuth } from '../context/AuthContext';
const LoginPage = () => {
// State for the form inputs
const [username, setUsername] = useState('');
const [password, setPassword] = useState('');
// State for handling errors from the API
const [error, setError] = useState('');
// Hook to programmatically navigate the user
const navigate = useNavigate();
const { login } = useAuth(); // Get the login function from the context

const [success, setSuccess] = useState('');
const [isSubmitting, setIsSubmitting] = useState(false);

const handleLogin = async (e) => {
// Prevent the default form submission which reloads the page
```

e.preventDefault();
setError(''); // Clear previous errors
setSuccess('');
setIsSubmitting(true);
try {
// Make the POST request to the login endpoint
const response = await api.post('/auth/login', {
username: username,
password: password,
});

```
    // Assuming the API returns a token in response.data.accessToken
    if (response.data && response.data.accessToken) {
      setSuccess('Login successful! Redirecting...'); // <-- 4. Set success
  message instead of alert

      // Use a timeout to give the user time to read the message before
  redirecting
      setTimeout(() => {
        login(response.data.accessToken);
        navigate('/');
      }, 1500); // 1.5 second delay
    }
  } catch (err) {
    // If the server responds with an error (e.g., 401 Unauthorized)
    console.error('Login failed:', err);
    setError('Invalid username or password. Please try again.');
    setIsSubmitting(false);
  }
```

};

return (

Login

Username:
<input
type="text"

```
id="username"
value={username}
onChange={(e) => setUsername(e.target.value)}
required
/>


Password:
<input
type="password"
id="password"
value={password}
onChange={(e) => setPassword(e.target.value)}
required
/>

{/ Display error message if it exists /}
{error &&

{error}

}
{success && <p style={{ color: 'green' }}>{success}

}
  {isSubmitting ? 'Logging in...' : 'Login'}



);
};

export default LoginPage;

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/pages/QuestionDetailPage.js =
import React, { useState, useEffect } from 'react';
import { useParams } from 'react-router-dom'; // Hook to get URL parameters
import api from '../services/api';
import { useAuth } from '../context/AuthContext';
import AnswerForm from '../components/AnswerForm';
import Vote from '../components/Vote';
```

// We'll reuse the AnswerResponse structure implicitly, or create a dedicated AnswerDisplay component later.

const QuestionDetailPage = () => {
// Get the question ID from the URL parameters (e.g., /questions/123)
const { id } = useParams();

// State for the question data, loading, and error
const [question, setQuestion] = useState(null);
const [loading, setLoading] = useState(true);
const [error, setError] = useState(null);
const { isLoggedIn } = useAuth(); // <-- 3. Get login status

// Fetch the question details when the component mounts or the ID changes
const fetchQuestionDetails = async () => {
setLoading(true); // Start loading
setError(null); // Clear previous errors

```
  try {
    // Fetch the specific question using its ID
    const response = await api.get(`/questions/${id}`);
    setQuestion(response.data);
  } catch (err) {
    console.error('Failed to fetch question details:', err);
    // Set a user-friendly error message
    setError('Could not load question. Please try again later.');
  } finally {
    setLoading(false); // Stop loading
  }
};
```

useEffect(() => {
fetchQuestionDetails();
}, [id]); // Re-run the effect if the ID in the URL changes

// Render loading or error states
if (loading) {
return

Loading question details...
;
}

if (error) {
return

Error: {error}
;
}
if (!question) return
Question not found.
;
// Render the question and its answers
return (

{/ --- *QUESTION SECTION* --- /}
{/ *The title and metadata are now outside the flex container* /}

{question.title}

<p style={{color: '#6a737c', fontSize: '13px'}}>
Asked by: {question.authorUsername} on {new Date(question.createdAt).toLocaleString()}

```
    {/* This new flex container aligns the vote component with the question
body */}
    <div style={{ display: 'flex', alignItems: 'flex-start' }}>
        <Vote post={question} postType="question" onVoteSuccess=
{fetchQuestionDetails} />
        <div className="question-detail-body" style={{ flex: 1 }}>
            {question.body}
        </div>
    </div>

    {/* --- ANSWERS SECTION --- */}
    <h3 style={{marginTop: '40px'}}>Answers ({question.answers.length})</h3>

    {question.answers.length > 0 ? (
        question.answers.map((answer) => (
            // We apply the same pattern to each answer
            <div key={answer.id} style={{ borderTop: '1px solid #d6d9dc',
paddingTop: '16px', marginTop: '16px' }}>
                <div style={{ display: 'flex', alignItems: 'flex-start' }}>
```

```
                    <Vote post={answer} postType="answer" onVoteSuccess=
{fetchQuestionDetails} />
                    <div className="question-detail-body" style={{ flex: 1
}}>
                        {answer.body}
                    </div>
                </div>
                <p style={{textAlign: 'right', color: '#6a737c', fontSize:
'13px', marginTop: '10px'}}>
                    Answered by: {answer.authorUsername} on {new
Date(answer.createdAt).toLocaleString()}
                </p>
            </div>
        ))
    ) : (
        <p>No answers yet.</p>
    )}

    {/* --- POST AN ANSWER SECTION (remains the same) --- */}
    {isLoggedIn ? (
        <AnswerForm questionId={id} onAnswerPosted={fetchQuestionDetails} />
    ) : (
        <p style={{ marginTop: '20px' }}>Please log in to post an answer.
</p>
    )}

</div>
```

);
};

export default QuestionDetailPage;

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/pages/AskQuestionPage.js =

```
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import api from '../services/api';

const AskQuestionPage = () => {
const [title, setTitle] = useState('');
const [body, setBody] = useState('');
const [error, setError] = useState('');
const navigate = useNavigate();
const [success, setSuccess] = useState(''); // <-- 1. Add state for success message
const [isSubmitting, setIsSubmitting] = useState(false); // <-- 2. Add state for submission status
```

const handleAskQuestion = async (e) => {
e.preventDefault();
setError('');
setSuccess('');
setIsSubmitting(true); // <-- 3. Set submitting to true

```
if (!title || !body) {
  setError('Title and body are required.');
  return;
}

try {
  // The JWT token is automatically added by our axios interceptor!
  await api.post('/questions', {
    title: title,
    body: body,
  });

  setSuccess('Question posted successfully!');
  // Use a timeout to give the user time to read the message before
redirecting
    setTimeout(() => {
       // Redirect to the homepage to see the new question
  navigate('/');
    }, 1500); // 1.5 second delay


} catch (err) {
  console.error('Failed to post question:', err);
  setError('Failed to post question. You may need to log in again.');
    setIsSubmitting(false);
}
```

};

return (

Ask a Public Question

Title

```
<input
type="text"
id="title"
value={title}
onChange={(e) => setTitle(e.target.value)}
placeholder="e.g. How do I implement a protected route in React?"
style={{ width: '80%' }}
/>


Body


<textarea
id="body"
value={body}
onChange={(e) => setBody(e.target.value)}
rows="10"
style={{ width: '80%' }}
/>

{error &&

{error}

}
{success && <p style={{ color: 'green' }}>{success}

}
  {isSubmitting ? 'Posting Question...' : 'Post Your Question'}



);
};

export default AskQuestionPage;
```

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/pages/HomePage.js =

```
import React, { useState,useEffect } from 'react';
```

import { useSearchParams } from 'react-router-dom';
import api from '../services/api';
import { Link } from 'react-router-dom';
const HomePage = () => {
//state to store list of questions,
const [questions, setQuestions] = useState([]);
//state to handle loading and error states
const [loading, setLoading] = useState(true);
const [error, setError] = useState(null);

```
//useEffect hook to fetch data when the component mounts
useEffect(()=>{
        const fetchQuestions = async () => {
            try{
                //make the api call using our api service instance
                const response = await api.get('/questions');
                //update the state with the data from the response
                setQuestions(response.data);
            }catch(err){
                //if an error occurs update error state
                setError('Failed to fetch questions. please try again
later');
                console.error(err);
            }finally{
                setLoading(false);
            }
        }
        fetchQuestions();
},[]);

if(loading) {
    return <div>Loading questions...</div>
}

if(error){
    return <div>Error: {error}</div>
}
```

return (

All Questions

{questions.length > 0 ? (
questions.map((question) => (

{/ *Wrap the title in a Link component* /}
<h3 style={{ margin: '0 0 5px 0' }}>
<Link to={ `/questions/${question.id}` }>{question.title}

<p style={{ margin: '0' }}>Asked by: {question.authorUsername}

))
) : (

No questions have been asked yet.

)}

);
};

export default HomePage;

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/pages/RegisterPage.js =
import React, { useState } from 'react';
import { useNavigate } from 'react-router-dom';
import api from '../services/api';

const RegisterPage = () => {
const [username, setUsername] = useState('');
const [email, setEmail] = useState('');
const [password, setPassword] = useState('');
const [error, setError] = useState('');
const navigate = useNavigate();
const [success, setSuccess] = useState(''); // <-- 1. Add state for success message
const [isSubmitting, setIsSubmitting] = useState(false); // <-- 2. Add state for submission status

const handleRegister = async (e) => {
e.preventDefault();

setError('');
setSuccess('');
setIsSubmitting(true); // <-- 3. Set submitting to true

```
  if (!username || !email || !password) {
      setError('All fields are required.');
      return;
  }

  try {
    await api.post('/auth/register', {
      username,
      email,
      password,
    });

    setSuccess('Registration successful! Please log in.');
    // Use a timeout to give the user time to read the message before
  redirecting
      setTimeout(() => {
          navigate('/login'); // Redirect to login page after successful
  registration
      }, 1500); // 1.5 second delay


  } catch (err) {
    console.error('Registration failed:', err);
    if (err.response && err.response.data) {
      // Use the error message from the backend if it exists
      setError(err.response.data);
    } else {
      setError('Registration failed. Please try again.');
    }
    setIsSubmitting(false); // <-- 5. Re-enable the form on failure
  }
```

};

return (

Create an Account

```
Username:
<input
type="text"
id="username"
value={username}
onChange={(e) => setUsername(e.target.value)}
required
/>


Email:
<input
type="email"
id="email"
value={email}
onChange={(e) => setEmail(e.target.value)}
required
/>


Password:
<input
type="password"
id="password"
value={password}
onChange={(e) => setPassword(e.target.value)}
required
minLength="8"
/>

{error &&

{error}

}
{success && <p style={{ color: 'green' }}>{success}

}
```

{isSubmitting ? 'Registering...' : 'Register'}

```
);
};

export default RegisterPage;

= /Users/dines/Documents/GitHub/stackunderflow-ui/src/services/api.js =
import axios from 'axios';

/// Get the backend URL from environment variables.
// This makes it easy to switch between local development and a live deployment.
// We'll fall back to localhost:8080 if the variable isn't set.
const API_URL = process.env.REACT_APP_API_URL || 'http://localhost:8080/api';

console.log(API_URL);
// Create an instance of axios with a custom configuration.
const api = axios.create({
baseURL: API_URL,
});

// ====================================================
// THIS IS THE NEW, IMPORTANT PART
// ====================================================
// Axios Request Interceptor
api.interceptors.request.use(
(config) => {
// Get the token from localStorage
const token = localStorage.getItem('token');
if (token) {
// If the token exists, add it to the Authorization header
config.headers.Authorization = `Bearer ${token}`;
}
return config;
},
(error) => {
// Handle request errors
return Promise.reject(error);
}
);
// ====================================================
```

// === //
// == NEW VOTE FUNCTIONS == //
// == //

/**

- Sends a vote for a specific question.
- @param {number} questionId The ID of the question to vote on.
- @param {string} voteType Can be 'UPVOTE' or 'DOWNVOTE'.
- @returns {Promise} The axios promise for the request.
  */
  export const voteOnQuestion = (questionId, voteType) => {
  return api.post(`/questions/${questionId}/vote` , { voteType });
  };

/**

- Sends a vote for a specific answer.
- @param {number} answerId The ID of the answer to vote on.
- @param {string} voteType Can be 'UPVOTE' or 'DOWNVOTE'.
- @returns {Promise} The axios promise for the request.
  */
  export const voteOnAnswer = (answerId, voteType) => {
  return api.post(`/answers/${answerId}/vote` , { voteType });
  };

export default api;