```
.
├── README.md
├── package-lock.json
├── package.json
├── public
│   ├── favicon.ico
│   ├── images
│   │   ├── blackboard.jpg
│   │   ├── frettolosa.jpg
│   │   ├── gesso.jpg
│   │   ├── graffiata.jpg
│   │   ├── lavata.jpg
│   │   ├── nera.jpg
│   │   ├── prodotti.json
│   │   └── verde.jpg
│   ├── index.html
│   ├── logo192.png
│   ├── logo512.png
│   ├── manifest.json
│   └── robots.txt
└── src
    ├── App.css
    ├── App.jsx
    ├── App.test.js
    ├── components
    │   ├── Auth
    │   │   ├── Login
    │   │   │   ├── Login.css
    │   │   │   └── Login.jsx
    │   │   └── Register
    │   │       ├── Register.css
    │   │       └── Register.jsx
    │   ├── Board
    │   │   ├── Board.css
    │   │   ├── Board.jsx
    │   │   └── actions
    │   │       ├── addItem.js
    │   │       ├── deleteItem.js
    │   │       ├── fetchItems.js
    │   │       ├── toggleBought.js
    │   │       └── updateItemName.js
    │   ├── Dashboard
    │   │   ├── Calendar
    │   │   │   ├── Calendar.css
    │   │   │   └── Calendar.jsx
    │   │   ├── Dashboard.css
    │   │   ├── Dashboard.jsx
    │   │   └── Preview
    │   │       ├── Preview.css
    │   │       └── Preview.jsx
    │   ├── ErrorBoundary
    │   │   └── ErrorBoundary.jsx
    │   ├── Header
    │   │   ├── Actions
```

```
│   │   │   ├── addBoard.js
│   │   │   ├── changeBackground.js
│   │   │   ├── deleteBoard.js
│   │   │   ├── fetchBoards.js
│   │   │   └── updateBoardName.js
│   │   ├── Header.css
│   │   └── Header.jsx
│   └── PrivateRoute
│       └── PrivateRoute.jsx
├── contexts
├── index.css
├── index.js
├── logo.svg
├── reportWebVitals.js
├── services
│   ├── api.js
│   ├── authService.js
│   ├── boardService.js
│   └── noteService.js
└── setupTests.js
```

```
alessandrotornabene@4f34e342-5886-4f1a-ba7d-072eedb502eb lavagnetta %
find src -type f \( -name "*.js" -o -name "*.jsx" -o -name "*.css" \)
-exec echo -e "\n\n==============================\nFile: {}
\n==============================\n" \; -exec cat {} \; -exec echo -e
"\n\n" \;

-e \n\n==============================\nFile: src/
reportWebVitals.js\n==============================\n
const reportWebVitals = onPerfEntry => {
  if (onPerfEntry && onPerfEntry instanceof Function) {
    import('web-vitals').then(({ getCLS, getFID, getFCP, getLCP, getTTFB
}) => {
      getCLS(onPerfEntry);
      getFID(onPerfEntry);
      getFCP(onPerfEntry);
      getLCP(onPerfEntry);
      getTTFB(onPerfEntry);
    });
  }
};

export default reportWebVitals;
-e \n\n
-e \n\n==============================\nFile: src/
App.css\n==============================\n
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}
```

```
@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
}

.App-link {
  color: #61dafb;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }

  to {
    transform: rotate(360deg);
  }
}

body {
  background-color: #f8f9fa;
}-e \n\n
-e \n\n==============================\nFile: src/
index.js\n==============================\n
// src/index.js

import React from 'react';
import ReactDOM from 'react-dom/client'; // Importa da 'react-dom/
client'
import './index.css';
import App from './App';


// Importazione di Bootstrap CSS
import 'bootstrap/dist/css/bootstrap.min.css';

// Importazione di Bootstrap Icons
import 'bootstrap-icons/font/bootstrap-icons.css';

// Importazione di Slick Carousel CSS
import 'slick-carousel/slick/slick.css';
import 'slick-carousel/slick/slick-theme.css';
```

```
// Importazione di React Toastify CSS (opzionale)
import 'react-toastify/dist/ReactToastify.css';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
-e \n\n
-e \n\n==============================\nFile: src/
index.css\n==============================\n
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;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
-e \n\n
-e \n\n==============================\nFile: src/components/
ErrorBoundary/ErrorBoundary.jsx\n==============================\n
// src/components/ErrorBoundary/ErrorBoundary.jsx

import React from 'react';

class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // Aggiorna lo stato per mostrare l'interfaccia di fallback
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {
    // Puoi registrare l'errore in un servizio di log
    console.error("Errore catturato da ErrorBoundary:", error,
errorInfo);
  }

  render() {
    if (this.state.hasError) {
      // Puoi renderizzare qualsiasi interfaccia di fallback
      return <h1>Qualcosa è andato storto.</h1>;
    }
```

```
    return this.props.children;
  }
}

export default ErrorBoundary;
-e \n\n
-e \n\n==============================\nFile: src/components/Board/
Board.jsx\n==============================\n
// src/components/Board/Board.jsx

import React, { useState, useEffect, useRef, useCallback } from 'react';
import './Board.css';

// Importa le funzioni dai servizi
import { fetchItems } from './actions/fetchItems';
import { addItem as addItemAction } from './actions/addItem';
import { deleteItem as deleteItemAction } from './actions/deleteItem';
import { updateItemName as updateItemNameAction } from './actions/
updateItemName';
import { toggleBought as toggleBoughtAction } from './actions/
toggleBought';
import { toast } from 'react-toastify';

const Board = ({ selectedBoard }) => {
  const [items, setItems] = useState([]);
  const [newItem, setNewItem] = useState('');
  const [editingIndex, setEditingIndex] = useState(null);
  const [editedName, setEditedName] = useState('');
  const [isLoading, setIsLoading] = useState(false);
  const inputRef = useRef(null);

  /**
   * Funzione per recuperare gli elementi della lavagnetta.
   * Utilizza useCallback per memorizzare la funzione e prevenire
ricreazioni non necessarie.
   */
  const fetchItemsCallback = useCallback(async () => {
    if (!selectedBoard || !selectedBoard.id) {
      toast.error('Lavagnetta selezionata non è valida.');
      return;
    }
    try {
      await fetchItems(selectedBoard, setItems, setIsLoading);
      // Se desideri utilizzare 'bought' nel frontend, puoi mappare
'is_purchased' a 'bought'
      // Altrimenti, usa direttamente 'is_purchased'
      /*
      setItems((prevItems) => prevItems.map(item => ({
        ...item,
        bought: item.is_purchased
      })));
      */
    } catch (error) {
      console.error('Errore nel fetchItemsCallback:', error);
    }
  }, [selectedBoard]);
```

```
  // Effettua il fetch degli elementi quando selectedBoard cambia
  useEffect(() => {
    if (selectedBoard) {
      fetchItemsCallback();
    } else {
      setItems([]);
    }
  }, [selectedBoard, fetchItemsCallback]);

  // Aggiungi un useEffect per monitorare gli aggiornamenti di items
  useEffect(() => {
    console.log('Items state updated:', items);
  }, [items]);

  // Funzione per aggiungere un nuovo elemento
  const handleAddItem = () => {
    if (!newItem.trim()) {
      toast.error('Inserisci un nome valido per il prodotto.');
      return;
    }
    addItemAction(newItem, items, setItems, setNewItem, selectedBoard)
      .then(() => {
        toast.success('Prodotto aggiunto con successo.');
      })
      .catch((error) => {
        console.error('Errore nell\'aggiunta del prodotto:', error);
        toast.error('Errore nell\'aggiunta del prodotto.');
      });
  };

  // Funzione per rimuovere un elemento
  const handleDeleteItem = (index) => {
    deleteItemAction(index, items, setItems, selectedBoard)
      .then(() => {
        toast.success('Prodotto eliminato con successo.');
      })
      .catch((error) => {
        console.error('Errore nell\'eliminazione del prodotto:', error);
        toast.error('Errore nell\'eliminazione del prodotto.');
      });
  };

  // Funzione per alternare lo stato "bought" di un elemento
  const handleToggleBought = (index) => {
    toggleBoughtAction(index, items, setItems, selectedBoard)
      .then(() => {
        toast.success("Stato dell'elemento aggiornato.");
      })
      .catch((error) => {
        console.error("Errore nell'aggiornamento dell'elemento:",
error);
        toast.error("Errore nell'aggiornamento dell'elemento.");
      });
  };
```

```
  // Funzione per aggiornare il nome di un elemento
  const handleUpdateItemName = (index) => {
    if (!editedName.trim()) {
      toast.error('Il nome del prodotto non può essere vuoto.');
      return;
    }
    updateItemNameAction(index, editedName, items, setItems,
setEditingIndex, setEditedName, selectedBoard)
      .then(() => {
        toast.success('Nome del prodotto aggiornato con successo.');
      })
      .catch((error) => {
        console.error('Errore nell\'aggiornamento del nome del
prodotto:', error);
        toast.error('Errore nell\'aggiornamento del nome del
prodotto.');
      });
  };

  if (!selectedBoard) {
    return <div className="board-container">Seleziona una lavagnetta per
visualizzarla.</div>;
  }

  // Costruisci l'URL dell'immagine di sfondo
  const backgroundImageUrl = `/images/${selectedBoard.background}`;

  return (
    <div
      className="board-container"
      style={{
        backgroundImage: `url(${backgroundImageUrl})`,
        backgroundSize: 'cover',
        backgroundPosition: 'center',
        minHeight: '100vh',
        paddingTop: '60px', // Altezza dell'header
        boxSizing: 'border-box',
        position: 'relative',
      }}
    >
      {/* Immagine di sfondo per gestire errori */}
      <img
        src={backgroundImageUrl}
        alt="Sfondo della Lavagnetta"
        style={{ display: 'none' }} // Nascondi l'immagine, ma usa
l'evento onError
        onError={(e) => {
          e.target.src = '/images/default-background.jpg'; // Immagine
di fallback
        }}
      />
      {isLoading && <div className="loading-overlay">Caricamento...</
div>}
      <div className="add-item">
        <input
          type="text"
```

```
          value={newItem}
          onChange={(e) => setNewItem(e.target.value)}
          placeholder="Aggiungi un prodotto"
          onKeyDown={(e) => {
            if (e.key === 'Enter') {
              handleAddItem();
            }
          }}
          ref={inputRef}
          aria-label="Aggiungi un prodotto"
        />
        <button onClick={handleAddItem} aria-label="Aggiungi">
          Aggiungi
        </button>
      </div>
      <div className="product-list-container">
        <ul className="item-list">
          {items.map((item, index) => (
            <li key={item.id || index} className="list-group-item"> {/*
Usa item.id se disponibile */}
              {editingIndex === index ? (
                <input
                  value={editedName}
                  onChange={(e) => setEditedName(e.target.value)}
                  onBlur={() => handleUpdateItemName(index)}
                  onKeyDown={(e) => {
                    if (e.key === 'Enter') {
                      handleUpdateItemName(index);
                    } else if (e.key === 'Escape') {
                      setEditingIndex(null);
                      setEditedName('');
                    }
                  }}
                  aria-label="Modifica elemento"
                />
              ) : (
                <span
                  className={item.is_purchased ? 'bought' : ''}
                  onClick={() => handleToggleBought(index)}
                  onDoubleClick={() => handleDeleteItem(index)}
                  onContextMenu={(e) => {
                    e.preventDefault();
                    setEditingIndex(index);
                    setEditedName(item.name);
                  }}
                  role="button"
                  tabIndex={0}
                  aria-label={`Elemento: ${item.name}`}
                >
                  {item.name}
                </span>
              )}
            </li>
          ))}
        </ul>
      </div>
```

```
    </div>
  );
};

export default Board;
-e \n\n
-e \n\n==============================\nFile: src/components/Board/
actions/addItem.js\n==============================\n
// src/components/Board/actions/addItem.js

import api from '../../../services/api';
import { toast } from 'react-toastify';

export const addItem = async (newItem, items, setItems, setNewItem,
selectedBoard) => {
    if (newItem.trim() === '') {
        toast.error("Il nome dell'elemento non può essere vuoto.");
        return;
    }
    try {
        const updatedItems = [...items, { name: newItem, bought:
false }];
        const response = await api.put(`/auth/boards/${selectedBoard.id}
`, {
            items: updatedItems,
        });
        setItems(response.data.items || updatedItems);
        setNewItem('');
        toast.success('Elemento aggiunto con successo.');
    } catch (error) {
        console.error("Errore nell'aggiunta dell'elemento:", error);
        toast.error("Errore nell'aggiunta dell'elemento.");
    }
};
-e \n\n
-e \n\n==============================\nFile: src/components/Board/
actions/toggleBought.js\n==============================\n
// src/components/Board/actions/toggleBought.js

import api from '../../../services/api';
import { toast } from 'react-toastify';

/**
 * Alterna lo stato di acquisto di un prodotto.
 * @param {number} index - Indice del prodotto nell'array.
 * @param {Array} items - Array dei prodotti.
 * @param {Function} setItems - Funzione per aggiornare lo stato degli
elementi.
 * @param {Object} selectedBoard - Lavagnetta selezionata.
 */
export const toggleBought = async (index, items, setItems,
selectedBoard) => {
    try {
        const item = items[index];
        if (!item) throw new Error('Elemento non trovato');
```

```
        // Inverti lo stato di 'is_purchased'
        const updatedItem = { ...item, is_purchased: !
item.is_purchased };
        const updatedItems = [...items];
        updatedItems[index] = updatedItem;

        // Aggiorna il backend
        await api.put(`/auth/boards/${selectedBoard.id}/products/$
{item.id}`, {
            is_purchased: updatedItem.is_purchased
        });

        // Aggiorna lo stato nel frontend
        setItems(updatedItems);
    } catch (error) {
        console.error('Errore nel toggleBought:', error);
        toast.error('Errore nell\'aggiornamento dello stato del
prodotto.');
        throw error;
    }
};
-e \n\n
-e \n\n============================\nFile: src/components/Board/
actions/fetchItems.js\n============================\n
// src/components/Board/actions/fetchItems.js

import api from '../../../services/api';
import { toast } from 'react-toastify';

/**
 * Recupera gli elementi della lavagnetta dal backend.
 * @param {Object} selectedBoard - Lavagnetta selezionata.
 * @param {Function} setItems - Funzione per impostare gli elementi
nello stato.
 * @param {Function} setIsLoading - Funzione per gestire lo stato di
caricamento.
 */
export const fetchItems = async (selectedBoard, setItems, setIsLoading)
=> {
    try {
        setIsLoading(true);
        const response = await api.get(`/auth/boards/$
{selectedBoard.id}/products`);

        // Verifica che la risposta abbia i prodotti
        if (response.data) {
            // Supponendo che la risposta sia un array di prodotti
            const items = response.data; // Assicurati che sia corretto

            // Se desideri utilizzare 'bought' nel frontend, puoi
mappare 'is_purchased' a 'bought'
            // Altrimenti, usa direttamente 'is_purchased'
            /*
            const mappedItems = items.map(item => ({
                ...item,
                bought: item.is_purchased
```

```
            }));
            setItems(mappedItems);
            */
            setItems(items);
        } else {
            setItems([]);
            toast.warning('Nessun prodotto trovato.');
        }
    } catch (error) {
        console.error('Errore nel recuperare gli elementi della
lavagnetta:', error);
        toast.error('Errore nel recuperare gli elementi della
lavagnetta.');
    } finally {
        setIsLoading(false);
    }
};
-e \n\n
-e \n\n============================\nFile: src/components/Board/
actions/deleteItem.js\n============================\n
// src/components/Board/actions/deleteItem.js

import api from '../../../services/api';
import { toast } from 'react-toastify';

export const deleteItem = async (index, items, setItems, selectedBoard)
=> {
    if (!window.confirm('Sei sicuro di voler eliminare questo
elemento?')) return;
    try {
        const updatedItems = items.filter((_, i) => i !== index);
        const response = await api.put(`/auth/boards/${selectedBoard.id}
`, {
            items: updatedItems,
        });
        setItems(response.data.items || updatedItems);
        toast.success('Elemento rimosso con successo.');
    } catch (error) {
        console.error("Errore nella rimozione dell'elemento:", error);
        toast.error("Errore nella rimozione dell'elemento.");
    }
};
-e \n\n
-e \n\n============================\nFile: src/components/Board/
actions/updateItemName.js\n============================\n
// src/components/Board/actions/updateItemName.js

import api from '../../../services/api';
import { toast } from 'react-toastify';

export const updateItemName = async (
    index,
    editedName,
    items,
    setItems,
    setEditingIndex,
```

```
    setEditedName,
    selectedBoard
) => {
    if (editedName.trim() === '') {
        toast.error("Il nome dell'elemento non può essere vuoto.");
        return;
    }
    try {
        const updatedItems = items.map((item, i) =>
            i === index ? { ...item, name: editedName } : item
        );
        const response = await api.put(`/auth/boards/${selectedBoard.id}
`, {
            items: updatedItems,
        });
        setItems(response.data.items || updatedItems);
        setEditingIndex(null);
        setEditedName('');
        toast.success("Nome dell'elemento aggiornato.");
    } catch (error) {
        console.error("Errore nell'aggiornamento dell'elemento:",
error);
        toast.error("Errore nell'aggiornamento dell'elemento.");
    }
};
-e \n\n
-e \n\n============================\nFile: src/components/Board/
Board.css\n============================\n
/* src/components/Board/Board.css */

/* Messaggio di Feedback */
.feedback-message {
    position: fixed;
    top: 20px;
    left: 50%;
    transform: translateX(-50%);
    padding: 10px 20px;
    border-radius: 5px;
    color: #fff;
    z-index: 1000;
    opacity: 0.9;
    transition: opacity 0.5s ease;
}

.feedback-success {
    background-color: #4caf50;
}

.feedback-error {
    background-color: #f44336;
}

/* Board Container */
.board-container {
    position: relative;
    width: 100%;
```

```
    min-height: 100vh;
    display: flex;
    flex-direction: column;
    padding: 20px;
    box-sizing: border-box;
    transition: margin-top 0.3s ease;
    margin-top: 60px;
    /* $header-height */
}

/* Contenitore della lista prodotti */
.product-list-container {
    position: relative;
    z-index: 1;
    pointer-events: auto;
    transition: transform 0.3s ease;
    overflow: hidden;
    flex: 1;
    /* Occupa lo spazio disponibile */
}

/* Abilita eventi sui figli */
.product-list-container * {
    pointer-events: auto;
}

/* Stile per il campo di input per aggiungere nuovi prodotti */
.add-item {
    position: relative;
    width: 100%;
    margin-bottom: 20px;
    z-index: 1006;
    pointer-events: auto;
    color: rgb(222, 11, 11);

    /* Cambiato da none a auto per permettere interazioni */
}

/* Wrapper per contenere input e suggerimenti */
.add-item .input-wrapper {
    position: relative;
    width: 100%;
    display: flex;
    align-items: center;
}

/* Elemento nascosto per supporto alla larghezza dinamica dell'input */
.add-item .mirror {
    position: absolute;
    top: 0;
    left: 0;
    visibility: hidden;
    white-space: pre;
    font-family: "Rock Salt", cursive;
    font-size: 1em;
    padding: 10px 0;
```

```
    border: none;
    box-sizing: border-box;
}

/* Stile per l'input di testo per aggiungere prodotti */
.add-item input {
    width: 100%;
    padding: 10px;
    border: none;
    border-radius: 0;
    font-family: "Rock Salt", cursive;
    background-color: transparent;
    color: rgb(222, 11, 11);
    transition: background-color 0.3s ease;
    font-size: 1em;
    position: relative;
    z-index: 1;
    pointer-events: auto;
}

.add-item input:focus {
    outline: none;
    background-color: rgba(255, 255, 255, 0.1);
}

.add-item input:not(:focus) {
    background-color: transparent;
}

.add-item input::placeholder {
    color: rgba(255, 255, 255, 0.6);
}

/* Stile per i suggerimenti che appaiono sotto l'input */
.add-item .suggestion {
    position: absolute;
    top: 10px;
    left: 0;
    color: rgba(255, 255, 255, 0.6);
    padding: 2px 5px;
    border-radius: 3px;
    pointer-events: auto;
    white-space: nowrap;
    overflow: hidden;
    text-overflow: ellipsis;
    font-family: "Rock Salt", cursive;
    font-size: 1em;
    z-index: 10;
    cursor: pointer;
    background-color: transparent;
}

.add-item .suggestion:hover {
    background-color: rgba(0, 0, 0, 0.7);
}
```

```
/* Lista dei prodotti */
.product-list-container .item-list {
    list-style: none;
    padding: 10px;
    margin: 0;
    overflow-y: auto;
    flex: 1;
    height: 100%;
    max-height: calc(100vh - 60px - 60px);
    /* $header-height + padding */
    scrollbar-width: thin;
    scrollbar-color: #888 transparent;
}

.product-list-container .item-list::after {
    content: "";
    display: block;
    height: 150px;
}

.product-list-container .item-list::-webkit-scrollbar {
    width: 8px;
}

.product-list-container .item-list::-webkit-scrollbar-track {
    background: transparent;
}

.product-list-container .item-list::-webkit-scrollbar-thumb {
    background-color: #888;
    border-radius: 4px;
    border: 2px solid transparent;
    background-clip: padding-box;
}

.product-list-container .item-list li {
    background-color: transparent;
    padding: 10px;
    margin-bottom: 5px;
    display: flex;
    justify-content: space-between;
    align-items: center;
    border-radius: 3px;
    border: none;
    /* Rimuove qualsiasi bordo eventualmente applicato */
    transition: background-color 0.3s ease;
    position: relative;
}

.product-list-container .item-list li:hover {
    background-color: rgba(255, 255, 255, 0.1);
}

.product-list-container .item-list li span.bought {
    text-decoration: line-through;
    color: rgba(255, 255, 255, 0.8);
```

```
    font-size: 0.8em;
    opacity: 0.7;
    transition: font-size 0.3s ease, opacity 0.3s ease;
}

.product-list-container .item-list li span {
    transition: color 0.3s ease;
    cursor: pointer;
    position: relative;
}

.product-list-container .item-list li input {
    width: 100%;
    padding: 8px;
    font-family: "Rock Salt", cursive;
    font-size: 1em;
    border: 1px solid #ccc;
    border-radius: 3px;
    outline: none;
}

.product-list-container .item-list li input:focus {
    border-color: #66afe9;
}-e \n\n
-e \n\n==============================\nFile: src/components/Auth/
Register/Register.css\n==============================\n
/* src/components/Auth/Register/Register.css */

/* Stili personalizzati per il componente Register */

/* Contenitore di registrazione */
.registration-container {
    max-width: 100%;
    margin: 20px auto;
    padding: 0 15px;
}

/* Stile della card di registrazione */
#registration-card {
    max-width: 500px;
    width: 100%;
    margin: 0 auto;
    border-radius: 20px;
    background: linear-gradient(145deg, #444, #222);
    box-shadow: 0 8px 20px rgba(0, 0, 0, 0.3);
    transition: transform 0.3s ease;
}

/* Effetto di hover sulla card */
#registration-card:hover {
    transform: scale(1.02);
}

/* Stile del titolo nella card */
#registration-card h2 {
    text-align: center;
```

```
    font-family: "Rock Salt", cursive;
    margin-bottom: 25px;
    font-size: 1.8em;
    color: #ffcc66;
    text-shadow: 1px 1px 5px rgba(0, 0, 0, 0.5);
}

/* Stili per i gruppi di form */
#registration-card form .form-group {
    margin-bottom: 15px;
    position: relative;
}

/* Stili per il gruppo di input */
#registration-card form .form-group .input-group {
    position: relative;
}

/* Stili per gli input */
#registration-card form .form-group .input-group .form-control {
    font-family: "Roboto", sans-serif;
    /* Cambiato il font a Roboto */
    background-color: #444;
    color: #fff;
    border: transparent;
    border-radius: 5px;
    padding-top: 20px;
    /* Spazio per il testo */
    padding-left: 45px;
    /* Spazio per l'icona a sinistra */
    padding-right: 45px;
    /* Spazio per l'icona a destra */
    text-align: center;
    transition: background-color 0.3s ease, border-color 0.3s ease,
        text-align 0.3s ease;
    position: relative;
    z-index: 1;
    /* Assicura che il campo sia sotto le icone */
}

/* Stili per l'input quando è in focus */
#registration-card form .form-group .input-group .form-control:focus {
    outline: none !important;
    background-color: transparent !important;
    border-color: transparent !important;
    box-shadow: none !important;
    text-align: left;
    /* Allinea a sinistra durante l'inserimento */
}

/* Stile delle etichette flottanti */
#registration-card form .form-group .input-group .floating-label {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
```

```
    font-size: 1em;
    color: #ffcc66;
    font-family: "Rock Salt", cursive;
    transition: all 0.2s ease;
    pointer-events: none;
    padding: 0 5px;
    z-index: 1;
    /* Assicura che l'etichetta sia sopra l'input ma sotto le icone */
}

/* Sposta l'etichetta sopra il campo quando è in focus o contiene testo
*/
#registration-card form .form-group .input-group .form-
control:not(:placeholder-shown)+.floating-label,
#registration-card form .form-group .input-group .form-
control:focus+.floating-label {
    top: -10px;
    left: 45px;
    transform: translate(0, 0);
    font-size: 0.6em;
    color: #ffaa33;
    text-align: left;
}

/* Stile delle icone all'interno degli input */
#registration-card form .form-group .input-group .input-group-text {
    position: absolute;
    top: 50%;
    transform: translateY(-50%);
    background-color: transparent !important;
    border: none;
    color: #ffcc66;
    font-size: 1.2em;
    z-index: 2;
    /* Assicura che le icone siano sopra l'input e l'etichetta */
}

/* Posizionamento delle icone a sinistra */
#registration-card form .form-group .input-group .input-group-text.icon-
left {
    left: 10px;
}

/* Stile delle icone di toggle password a destra */
#registration-card form .form-group .input-group .input-group-
text.toggle-password {
    right: 10px;
    cursor: pointer;
}

/* Effetto hover sulle icone di toggle password */
#registration-card form .form-group .input-group .input-group-
text.toggle-password:hover {
    color: #ffaa33;
}
```

```
/* Stile dei messaggi di errore */
#registration-card .error-message {
    color: #ff6b6b;
    font-size: 0.9em;
    font-style: italic;
}

/* Stile del bottone di registrazione */
#registration-card button {
    background-color: #ffcc66;
    color: #333;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease, transform 0.2s ease;
    font-family: "Rock Salt", cursive;
    font-size: 1em;
    box-shadow: 0 4px 15px rgba(0, 0, 0, 0.3);
}

/* Effetto hover sul bottone di registrazione */
#registration-card button:hover {
    background-color: #ffaa33;
    transform: scale(1.1);
}

/* Effetto active sul bottone di registrazione */
#registration-card button:active {
    transform: scale(0.95);
}

/* Stile del paragrafo e dei link */
#registration-card p {
    font-size: 0.95em;
    color: #fff;
    /* Cambia il colore della frase in bianco */
}

#registration-card p a {
    color: #ffcc66;
    text-decoration: underline;
    font-family: "Rock Salt", cursive;
    transition: color 0.3s ease;
}

#registration-card p a:hover {
    color: #ffaa33;
}

/* Media Queries per la Responsività */
@media (max-width: 768px) {
    #registration-card {
        max-width: 90%;
        padding: 20px;
        margin: 10px auto;
    }
```

```
    #registration-card h2 {
        font-size: 1.5em;
        margin-bottom: 15px;
    }

    #registration-card button {
        width: 100%;
    }
}

@media (max-width: 576px) {
    #registration-card {
        padding: 15px;
    }

    #registration-card h2 {
        font-size: 1.4em;
    }

    #registration-card button {
        padding: 10px;
        font-size: 0.9em;
    }
}-e \n\n
-e \n\n==============================\nFile: src/components/Auth/
Register/Register.jsx\n==============================\n
// src/components/Auth/Register/Register.jsx

import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import './Register.css';
import authService from '../../../services/authService';

const Register = () => {
  const navigate = useNavigate();

  const [formData, setFormData] = useState({
    username: '',
    email: '',
    password: '',
    confirmPassword: '',
  });

  const [errors, setErrors] = useState({
    username: '',
    email: '',
    password: '',
    confirmPassword: '',
    general: '',
  });

  const [showPassword, setShowPassword] = useState(false);
  const [showConfirmPassword, setShowConfirmPassword] = useState(false);

  const handleChange = (e) => {
```

```
  const { name, value } = e.target;

  setFormData((prevData) => ({
    ...prevData,
    [name]: value,
  }));

  setErrors((prevErrors) => ({
    ...prevErrors,
    [name]: '',
    general: '',
  }));
};

const validate = () => {
  let valid = true;
  let newErrors = {
    username: '',
    email: '',
    password: '',
    confirmPassword: '',
    general: '',
  };

  if (!formData.username.trim()) {
    newErrors.username = 'Nome utente è obbligatorio.';
    valid = false;
  }

  if (!formData.email) {
    newErrors.email = 'Email è obbligatoria.';
    valid = false;
  } else if (!/\S+@\S+\.\S+/.test(formData.email)) {
    newErrors.email = 'Inserisci un\'email valida.';
    valid = false;
  }

  if (!formData.password) {
    newErrors.password = 'Password è obbligatoria.';
    valid = false;
  }

  if (!formData.confirmPassword) {
    newErrors.confirmPassword = 'Conferma Password è obbligatoria.';
    valid = false;
  } else if (formData.password !== formData.confirmPassword) {
    newErrors.confirmPassword = 'Le password non corrispondono.';
    valid = false;
  }

  setErrors(newErrors);
  return valid;
};

const handleSubmit = async (e) => {
  e.preventDefault();
```

```
    if (!validate()) {
      setErrors((prevErrors) => ({
        ...prevErrors,
        general: 'Per favore, completa tutti i campi correttamente.',
      }));
      return;
    }

    try {
      const response = await authService.register(formData.email,
formData.password, formData.username);

      if (response.token) {
        navigate('/auth/login');
      }
    } catch (error) {
      console.error('Errore durante la registrazione:', error);
      if (error.response && error.response.status === 400) {
        setErrors((prevErrors) => ({
          ...prevErrors,
          general: error.response.data.error || 'Errore durante la
registrazione. Per favore, riprova.',
        }));
      } else {
        setErrors((prevErrors) => ({
          ...prevErrors,
          general: 'Errore durante la registrazione. Per favore,
riprova.',
        }));
      }
    }
  };

  const togglePasswordVisibility = () => {
    setShowPassword((prev) => !prev);
  };

  const toggleConfirmPasswordVisibility = () => {
    setShowConfirmPassword((prev) => !prev);
  };

  return (
    <div className="auth-container container-fluid d-flex justify-
content-center align-items-center min-vh-100">
      <div className="card p-4 shadow-lg" id="auth-card">
        <h2 className="text-center mb-4">Crea il tuo account</h2>
        <form onSubmit={handleSubmit} className="needs-validation"
noValidate>
          <div className="form-group mb-3 position-relative">
            <div className="input-group">
              <span className="input-group-text icon-left">
                <i className="bi bi-person"></i>
              </span>
              <input
                type="text"
```

```
                name="username"
                value={formData.username}
                onChange={handleChange}
                className="form-control text-center"
                placeholder=" "
                required
              />
              <label htmlFor="username" className="floating-label">Nome
Utente</label>
            </div>
            {errors.username && (
              <div className="text-danger text-center">
                {errors.username}
              </div>
            )}
          </div>

          <div className="form-group mb-3 position-relative">
            <div className="input-group">
              <span className="input-group-text icon-left">
                <i className="bi bi-envelope"></i>
              </span>
              <input
                type="email"
                name="email"
                value={formData.email}
                onChange={handleChange}
                className="form-control text-center"
                placeholder=" "
                required
              />
              <label htmlFor="email" className="floating-label">Email</
label>
            </div>
            {errors.email && (
              <div className="text-danger text-center">
                {errors.email}
              </div>
            )}
          </div>

          <div className="form-group mb-3 position-relative">
            <div className="input-group">
              <span className="input-group-text icon-left">
                <i className="bi bi-lock"></i>
              </span>
              <input
                type={showPassword ? 'text' : 'password'}
                name="password"
                value={formData.password}
                onChange={handleChange}
                className="form-control text-center"
                placeholder=" "
                required
              />
```

```
              <label htmlFor="password" className="floating-
label">Password</label>
              <span
                className="input-group-text toggle-password"
                onClick={togglePasswordVisibility}
                style={{ cursor: 'pointer' }}
              >
                <i className={`bi ${showPassword ? 'bi-eye-slash' : 'bi-
eye'}`}></i>
              </span>
            </div>
            {errors.password && (
              <div className="text-danger text-center">
                {errors.password}
              </div>
            )}
          </div>

          <div className="form-group mb-3 position-relative">
            <div className="input-group">
              <span className="input-group-text icon-left">
                <i className="bi bi-lock-fill"></i>
              </span>
              <input
                type={showConfirmPassword ? 'text' : 'password'}
                name="confirmPassword"
                value={formData.confirmPassword}
                onChange={handleChange}
                className="form-control text-center"
                placeholder=" "
                required
              />
              <label htmlFor="confirmPassword" className="floating-
label">Conferma Password</label>
              <span
                className="input-group-text toggle-password"
                onClick={toggleConfirmPasswordVisibility}
                style={{ cursor: 'pointer' }}
              >
                <i className={`bi ${showConfirmPassword ? 'bi-eye-slash'
: 'bi-eye'}`}></i>
              </span>
            </div>
            {errors.confirmPassword && (
              <div className="text-danger text-center">
                {errors.confirmPassword}
              </div>
            )}
          </div>

          {errors.general && (
            <div className="error-message mb-3 text-center text-danger">
              {errors.general}
            </div>
          )}
```

```
          <button
            type="submit"
            className={`btn btn-warning w-100 ${formData.username === ''
|| formData.email === '' || formData.password === '' ||
formData.confirmPassword === '' || errors.username || errors.email ||
errors.password || errors.confirmPassword ? 'opacity-50 cursor-not-
allowed' : ''}`}
            disabled={
              formData.username === '' ||
              formData.email === '' ||
              formData.password === '' ||
              formData.confirmPassword === '' ||
              errors.username ||
              errors.email ||
              errors.password ||
              errors.confirmPassword
            }
          >
            Registrati
          </button>
        </form>

        <p className="text-center mt-3">
          Hai già un account?
          <Link to="/auth/login" className="text-warning"> Accedi</Link>
        </p>
      </div>
    </div>
  );
};

export default Register;
-e \n\n
-e \n\n==============================\nFile: src/components/Auth/Login/
Login.css\n==============================\n
/* src/components/Auth/Login/Login.css */

/* Stili personalizzati per il componente Login */

/* Contenitore di autenticazione */
.auth-container {
    max-width: 100%;
    margin: 20px auto;
    padding: 0 15px;
}

/* Stile della card di autenticazione */
#auth-card {
    max-width: 500px;
    width: 100%;
    margin: 0 auto;
    border-radius: 20px;
    background: linear-gradient(145deg, #444, #222);
    box-shadow: 0 8px 20px rgba(0, 0, 0, 0.3);
    transition: transform 0.3s ease;
}
```

```
#auth-card:hover {
    transform: scale(1.02);
}

#auth-card h2 {
    text-align: center;
    font-family: "Rock Salt", cursive;
    margin-bottom: 25px;
    font-size: 1.8em;
    color: #ffcc66;
    text-shadow: 1px 1px 5px rgba(0, 0, 0, 0.5);
}

#auth-card form .form-group {
    margin-bottom: 15px;
    position: relative;
}

#auth-card form .form-group .input-group {
    position: relative;
}

#auth-card form .form-group .input-group .form-control {
    font-family: "Roboto", sans-serif;
    /* Cambiato il font a Roboto */
    background-color: #444;
    color: #fff;
    border: transparent;
    border-radius: 5px;
    padding-top: 20px;
    /* Spazio per il testo */
    padding-left: 45px;
    /* Spazio per l'icona a sinistra */
    padding-right: 45px;
    /* Spazio per l'icona a destra */
    text-align: center;
    transition: background-color 0.3s ease, border-color 0.3s ease,
        text-align 0.3s ease;
    position: relative;
    z-index: 1;
    /* Assicura che il campo sia sotto le icone */
}

#auth-card form .form-group .input-group .form-control:focus {
    outline: none !important;
    background-color: transparent !important;
    border-color: transparent !important;
    box-shadow: none !important;
    text-align: left;
    /* Allinea a sinistra durante l'inserimento */
}

#auth-card form .form-group .input-group .floating-label {
    position: absolute;
    top: 50%;
```

```
    left: 50%;
    transform: translate(-50%, -50%);
    font-size: 1em;
    color: #ffcc66;
    font-family: "Rock Salt", cursive;
    transition: all 0.2s ease;
    pointer-events: none;
    padding: 0 5px;
    z-index: 1;
    /* Assicura che l'etichetta sia sopra l'input ma sotto le icone */
}

/* Sposta l'etichetta sopra il campo quando è in focus o contiene testo
*/
#auth-card form .form-group .input-group .form-control:not(:placeholder-
shown)+.floating-label,
#auth-card form .form-group .input-group .form-control:focus+.floating-
label {
    top: -10px;
    left: 45px;
    transform: translate(0, 0);
    font-size: 0.6em;
    color: #ffaa33;
    text-align: left;
}

#auth-card form .form-group .input-group .input-group-text {
    position: absolute;
    top: 50%;
    transform: translateY(-50%);
    background-color: transparent !important;
    border: none;
    color: #ffcc66;
    font-size: 1.2em;
    z-index: 2;
    /* Assicura che le icone siano sopra l'input e l'etichetta */
}

#auth-card form .form-group .input-group .input-group-text.icon-left {
    left: 10px;
}

#auth-card form .form-group .input-group .input-group-text.toggle-
password {
    right: 10px;
    cursor: pointer;
}

#auth-card form .form-group .input-group .input-group-text.toggle-
password:hover {
    color: #ffaa33;
}

#auth-card .error-message {
    color: #ff6b6b;
    font-size: 0.9em;
```

```
    font-style: italic;
}

#auth-card button {
    background-color: #ffcc66;
    color: #333;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease, transform 0.2s ease;
    font-family: "Rock Salt", cursive;
    font-size: 1em;
    box-shadow: 0 4px 15px rgba(0, 0, 0, 0.3);
}

#auth-card button:hover {
    background-color: #ffaa33;
    transform: scale(1.1);
}

#auth-card button:active {
    transform: scale(0.95);
}

#auth-card p {
    font-size: 0.95em;
    color: #fff;
    /* Cambia il colore della frase in bianco */
}

#auth-card p a {
    color: #ffcc66;
    text-decoration: underline;
    font-family: "Rock Salt", cursive;
    transition: color 0.3s ease;
}

#auth-card p a:hover {
    color: #ffaa33;
}

/* Media Queries per la Responsività */
@media (max-width: 768px) {
    #auth-card {
        max-width: 90%;
        padding: 20px;
        margin: 10px auto;
    }

    #auth-card h2 {
        font-size: 1.5em;
        margin-bottom: 15px;
    }

    #auth-card button {
        width: 100%;
```

```
    }
}

@media (max-width: 576px) {
    #auth-card {
        padding: 15px;
    }

    #auth-card h2 {
        font-size: 1.4em;
    }

    #auth-card button {
        padding: 10px;
        font-size: 0.9em;
    }
}-e \n\n
-e \n\n==============================\nFile: src/components/Auth/Login/
Login.jsx\n==============================\n
// src/components/Auth/Login/Login.jsx

import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import './Login.css';
import authService from '../../../services/authService';

const Login = ({ setAuthenticated }) => {
  const navigate = useNavigate();

  // Stato per i dati del form
  const [formData, setFormData] = useState({
    email: '',
    password: '',
  });

  // Stato per la gestione degli errori
  const [errors, setErrors] = useState({
    email: '',
    password: '',
    general: '',
  });

  // Stato per la visibilità della password
  const [showPassword, setShowPassword] = useState(false);

  // Gestione delle modifiche negli input
  const handleChange = (e) => {
    const { name, value } = e.target;

    setFormData((prevData) => ({
      ...prevData,
      [name]: value,
    }));

    // Resetta l'errore del campo specifico
    setErrors((prevErrors) => ({
```

```
      ...prevErrors,
      [name]: '',
      general: '',
    }));
  };

  // Validazione del form
  const validate = () => {
    let valid = true;
    let newErrors = {
      email: '',
      password: '',
      general: '',
    };

    if (!formData.email) {
      newErrors.email = 'Email è obbligatoria.';
      valid = false;
    } else if (!/\S+@\S+\.\S+/.test(formData.email)) {
      newErrors.email = 'Inserisci un\'email valida.';
      valid = false;
    }

    if (!formData.password) {
      newErrors.password = 'Password è obbligatoria.';
      valid = false;
    }

    setErrors(newErrors);
    return valid;
  };

  // Gestione dell'invio del form
  const handleSubmit = async (e) => {
    e.preventDefault();

    if (!validate()) {
      setErrors((prevErrors) => ({
        ...prevErrors,
        general: 'Per favore, completa tutti i campi correttamente.',
      }));
      return;
    }

    try {
      const response = await authService.login(formData.email,
formData.password);

      if (response.token) {
        // Salva il token in localStorage
        localStorage.setItem('token', response.token);

        // Aggiorna lo stato di autenticazione
        setAuthenticated(true);

        // Login riuscito, reindirizza alla dashboard
```

```
        navigate('/dashboard');
      }
    } catch (error) {
      console.error('Errore durante il login:', error);
      if (error.response && error.response.status === 401) {
        setErrors((prevErrors) => ({
          ...prevErrors,
          general: 'Credenziali non valide.',
        }));
      } else {
        setErrors((prevErrors) => ({
          ...prevErrors,
          general: 'Errore durante il login. Riprova più tardi.',
        }));
      }
    }
  };

  // Funzione per togglare la visibilità della password
  const togglePasswordVisibility = () => {
    setShowPassword((prev) => !prev);
  };

  return (
    <div className="auth-container container-fluid d-flex justify-
content-center align-items-center min-vh-100">
      <div className="card p-4 shadow-lg" id="auth-card">
        <h2 className="text-center mb-4">Accedi</h2>
        <form onSubmit={handleSubmit} className="needs-validation"
noValidate>
          {/* Campo Email */}
          <div className="form-group mb-3 position-relative">
            <div className="input-group">
              <span className="input-group-text icon-left">
                <i className="bi bi-envelope"></i>
              </span>
              <input
                type="email"
                name="email"
                value={formData.email}
                onChange={handleChange}
                className="form-control text-center"
                placeholder=" "
                required
              />
              <label htmlFor="email" className="floating-label">Email</
label>
            </div>
            {errors.email && (
              <div className="text-danger text-center">
                {errors.email}
              </div>
            )}
          </div>

          {/* Campo Password */}
```

```
          <div className="form-group mb-3 position-relative">
            <div className="input-group">
              <span className="input-group-text icon-left">
                <i className="bi bi-lock"></i>
              </span>
              <input
                type={showPassword ? 'text' : 'password'}
                name="password"
                value={formData.password}
                onChange={handleChange}
                className="form-control text-center"
                placeholder=" "
                required
              />
              <label htmlFor="password" className="floating-
label">Password</label>
              <span
                className="input-group-text toggle-password"
                onClick={togglePasswordVisibility}
                style={{ cursor: 'pointer' }}
              >
                <i className={`bi ${showPassword ? 'bi-eye-slash' : 'bi-
eye'}`}></i>
              </span>
            </div>
            {errors.password && (
              <div className="text-danger text-center">
                {errors.password}
              </div>
            )}
          </div>

          {/* Messaggio di errore generale */}
          {errors.general && (
            <div className="error-message mb-3 text-center text-danger">
              {errors.general}
            </div>
          )}

          {/* Bottone di Accesso */}
          <button
            type="submit"
            className={`btn btn-warning w-100 ${formData.email === '' ||
formData.password === '' || errors.email || errors.password ?
'opacity-50 cursor-not-allowed' : ''}`}
            disabled={
              formData.email === '' ||
              formData.password === '' ||
              errors.email ||
              errors.password
            }
          >
            Accedi
          </button>
        </form>
```

```
        {/* Link alla Pagina di Registrazione */}
        <p className="text-center mt-3">
          Non hai un account?
          <Link to="/auth/register" className="text-warning">
Registrati</Link>
        </p>
      </div>
    </div>
  );
};

export default Login;
-e \n\n
-e \n\n============================\nFile: src/components/Dashboard/
Calendar/Calendar.jsx\n============================\n
// src/components/Dashboard/Calendar/Calendar.jsx

import React, { useState, useEffect, useCallback } from 'react';
import FullCalendar from '@fullcalendar/react';
import dayGridPlugin from '@fullcalendar/daygrid';
import interactionPlugin from '@fullcalendar/interaction';
import api from '../../../services/api';
import { Modal, Button, Form } from 'react-bootstrap';
import { toast } from 'react-toastify';
import './Calendar.css';

const CalendarComponent = () => {
    const [events, setEvents] = useState([]);
    const [showCreateModal, setShowCreateModal] = useState(false);
    const [showViewModal, setShowViewModal] = useState(false);
    const [showEditModal, setShowEditModal] = useState(false);
    const [currentNote, setCurrentNote] = useState({ id: '', title: '',
content: '', date: '' });
    const [noteToDelete, setNoteToDelete] = useState(null);
    const [showDeleteModal, setShowDeleteModal] = useState(false);

    // Definire fetchNotes usando useCallback per evitare il warning di
ESLint
    const fetchNotes = useCallback(async () => {
        try {
            const response = await api.get('/auth/notes');
            const notes = response.data.notes || [];
            console.log('Notes ricevute:', notes);
            const formattedEvents = notes.map(note => ({
                id: note.id.toString(), // Assicurati che l'ID sia una
stringa
                title: note.title,
                start: formatDate(note.date),
                allDay: true, // Tratta come evento di tutto il giorno
                extendedProps: {
                    content: note.content,
                },
            }));
            setEvents(formattedEvents);
            console.log('Events set:', formattedEvents);
        } catch (error) {
```

```
            console.error('Errore nel recuperare le note:', error);
            toast.error('Errore nel recuperare le note.');
        }
    }, []);

    useEffect(() => {
        fetchNotes();
    }, [fetchNotes]);

    const handleDateClick = (arg) => {
        setCurrentNote({ id: '', title: '', content: '', date:
arg.dateStr });
        setShowCreateModal(true);
    };

    const handleCloseCreateModal = () => setShowCreateModal(false);
    const handleCloseViewModal = () => setShowViewModal(false);
    const handleCloseEditModal = () => setShowEditModal(false);

    const handleInputChange = (e) => {
        const { name, value } = e.target;
        setCurrentNote(prev => ({
            ...prev,
            [name]: value,
        }));
    };

    const formatDate = (dateString) => {
        const date = new Date(dateString);
        const year = date.getFullYear();
        const month = (`0${date.getMonth() + 1}`).slice(-2);
        const day = (`0${date.getDate()}`).slice(-2);
        return `${year}-${month}-${day}`;
    };

    const handleCreateNote = async () => {
        const { title, content, date } = currentNote;
        if (!title || !date) {
            toast.warning('Titolo e data sono obbligatori.');
            return;
        }

        try {
            const response = await api.post('/auth/notes', { title,
content, date });
            const createdNote = response.data.note;
            console.log('Nota creata:', createdNote);

            const newEvent = {
                id: createdNote.id.toString(),
                title: createdNote.title,
                start: formatDate(createdNote.date),
                allDay: true, // Tratta come evento di tutto il giorno
                extendedProps: {
                    content: createdNote.content,
                },
```

```
            };
            setEvents(prevEvents => [...prevEvents, newEvent]);
            console.log('Events after addition:', [...events,
newEvent]);
            setShowCreateModal(false);
            setCurrentNote({ id: '', title: '', content: '', date:
'' });
            toast.success('Nota creata con successo!');
        } catch (error) {
            console.error('Errore nella creazione della nota:', error);
            toast.error('Errore nella creazione della nota.');
        }
    };

    const deleteNote = async (noteId) => {
        try {
            console.log(`Eliminando nota con ID: ${noteId} (tipo: $
{typeof noteId})`);
            const response = await api.delete(`/auth/notes/${noteId}`);
            console.log('Delete API response:', response.data);

            // Filtra manualmente l'evento dallo stato
            setEvents(prevEvents => prevEvents.filter(event =>
event.id !== noteId.toString()));
            console.log('Events after deletion:', events.filter(event =>
event.id !== noteId.toString()));

            toast.success('Nota eliminata con successo!');
        } catch (error) {
            console.error("Errore nell'eliminazione della nota:",
error);
            toast.error("Errore nell'eliminazione della nota.");
        }
    };

    const handleEventClick = (clickInfo) => {
        const { id, title, extendedProps } = clickInfo.event;
        setCurrentNote({
            id,
            title,
            content: extendedProps.content,
            date: formatDate(clickInfo.event.startStr)
        });
        setShowViewModal(true);
    };

    const handleConfirmDelete = () => {
        if (noteToDelete) {
            deleteNote(noteToDelete.id);
            setShowDeleteModal(false);
            setNoteToDelete(null);
        }
    };

    const handleCancelDelete = () => {
        setShowDeleteModal(false);
```

```
        setNoteToDelete(null);
    };

    const handleUpdateNote = async () => {
        const { id, title, content, date } = currentNote;
        if (!title || !date) {
            toast.warning('Titolo e data sono obbligatori.');
            return;
        }

        try {
            const response = await api.put(`/auth/notes/${id}`, { title,
content, date });
            const updatedNote = response.data.note;
            console.log('Nota aggiornata:', updatedNote);

            const updatedEvent = {
                id: updatedNote.id.toString(),
                title: updatedNote.title,
                start: formatDate(updatedNote.date),
                allDay: true, // Tratta come evento di tutto il giorno
                extendedProps: {
                    content: updatedNote.content,
                },
            };
            setEvents(prevEvents => prevEvents.map(event => event.id ===
id ? updatedEvent : event));
            console.log('Events after update:', events.map(event =>
event.id === id ? updatedEvent : event));
            setShowEditModal(false);
            setCurrentNote({ id: '', title: '', content: '', date:
'' });
            toast.success('Nota aggiornata con successo!');
        } catch (error) {
            console.error('Errore nell\'aggiornamento della nota:',
error);
            toast.error('Errore nell\'aggiornamento della nota.');
        }
    };

    const handleDeleteFromEdit = () => {
        setNoteToDelete({ id: currentNote.id, title:
currentNote.title });
        setShowDeleteModal(true);
        setShowEditModal(false);
    };

    const handleEditFromView = () => {
        setShowViewModal(false);
        setShowEditModal(true);
    };

    const handleDeleteFromView = () => {
        setNoteToDelete({ id: currentNote.id, title:
currentNote.title });
        setShowDeleteModal(true);
```

```
        setShowViewModal(false);
    };

    return (
        <div className="calendar-container">
            <FullCalendar
                plugins={[dayGridPlugin, interactionPlugin]}
                initialView="dayGridMonth"
                events={events}
                dateClick={handleDateClick}
                eventClick={handleEventClick}
                height="auto"
                timeZone="local" // Imposta la timezone locale
            />

            {/* Modal per Creazione di una Nuova Nota */}
            <Modal show={showCreateModal}
onHide={handleCloseCreateModal}>
                <Modal.Header closeButton>
                    <Modal.Title>Aggiungi una Nuova Nota</Modal.Title>
                </Modal.Header>
                <Modal.Body>
                    <Form>
                        <Form.Group controlId="noteTitle"
className="mb-3">
                            <Form.Label>Titolo</Form.Label>
                            <Form.Control
                                type="text"
                                name="title"
                                value={currentNote.title}
                                onChange={handleInputChange}
                                placeholder="Inserisci il titolo della
nota"
                                required
                            />
                        </Form.Group>
                        <Form.Group controlId="noteContent"
className="mb-3">
                            <Form.Label>Descrizione</Form.Label>
                            <Form.Control
                                as="textarea"
                                rows={3}
                                name="content"
                                value={currentNote.content}
                                onChange={handleInputChange}
                                placeholder="Inserisci una descrizione
(opzionale)"
                            />
                        </Form.Group>
                        <Form.Group controlId="noteDate"
className="mb-3">
                            <Form.Label>Data</Form.Label>
                            <Form.Control
                                type="date"
                                name="date"
                                value={currentNote.date}
```

```
                                onChange={handleInputChange}
                                required
                            />
                        </Form.Group>
                    </Form>
                </Modal.Body>
                <Modal.Footer>
                    <Button variant="secondary"
onClick={handleCloseCreateModal}>
                        Annulla
                    </Button>
                    <Button variant="primary" onClick={handleCreateNote}
>
                        Crea
                    </Button>
                </Modal.Footer>
            </Modal>

            {/* Modal per Visualizzare una Nota */}
            <Modal show={showViewModal} onHide={handleCloseViewModal}>
                <Modal.Header closeButton>
                    <Modal.Title>Visualizza Nota</Modal.Title>
                </Modal.Header>
                <Modal.Body>
                    <h5>{currentNote.title}</h5>
                    <p>{currentNote.content || 'Nessuna descrizione
fornita.'}</p>
                    <p><strong>Data:</strong> {currentNote.date}</p>
                </Modal.Body>
                <Modal.Footer>
                    <Button variant="danger"
onClick={handleDeleteFromView}>
                        Elimina
                    </Button>
                    <Button variant="secondary"
onClick={handleCloseViewModal}>
                        Chiudi
                    </Button>
                    <Button variant="primary"
onClick={handleEditFromView}>
                        Modifica
                    </Button>
                </Modal.Footer>
            </Modal>

            {/* Modal per Modifica di una Nota Esistente */}
            <Modal show={showEditModal} onHide={handleCloseEditModal}>
                <Modal.Header closeButton>
                    <Modal.Title>Modifica Nota</Modal.Title>
                </Modal.Header>
                <Modal.Body>
                    <Form>
                        <Form.Group controlId="editNoteTitle"
className="mb-3">
                            <Form.Label>Titolo</Form.Label>
                            <Form.Control
```

```
                                type="text"
                                name="title"
                                value={currentNote.title}
                                onChange={handleInputChange}
                                placeholder="Modifica il titolo della
nota"
                                required
                            />
                        </Form.Group>
                        <Form.Group controlId="editNoteContent"
classNam="mb-3">
                            <Form.Label>Descrizione</Form.Label>
                            <Form.Control
                                as="textarea"
                                rows={3}
                                name="content"
                                value={currentNote.content}
                                onChange={handleInputChange}
                                placeholder="Modifica la descrizione
(opzionale)"
                            />
                        </Form.Group>
                        <Form.Group controlId="editNoteDate"
classNam="mb-3">
                            <Form.Label>Data</Form.Label>
                            <Form.Control
                                type="date"
                                name="date"
                                value={currentNote.date}
                                onChange={handleInputChange}
                                required
                            />
                        </Form.Group>
                    </Form>
                </Modal.Body>
                <Modal.Footer>
                    <Button variant="danger"
onClick={handleDeleteFromEdit}>
                        Elimina
                    </Button>
                    <Button variant="secondary"
onClick={handleCloseEditModal}>
                        Annulla
                    </Button>
                    <Button variant="primary" onClick={handleUpdateNote}
>
                        Salva Modifiche
                    </Button>
                </Modal.Footer>
            </Modal>

            {/* Modal per Conferma Eliminazione Nota */}
            <Modal show={showDeleteModal} onHide={handleCancelDelete}>
                <Modal.Header closeButton>
                    <Modal.Title>Conferma Eliminazione</Modal.Title>
                </Modal.Header>
```

```
                <Modal.Body>
                    Sei sicuro di voler eliminare la nota:
<strong>{noteToDelete?.title}</strong>?
                </Modal.Body>
                <Modal.Footer>
                    <Button variant="secondary"
onClick={handleCancelDelete}>
                        Annulla
                    </Button>
                    <Button variant="danger"
onClick={handleConfirmDelete}>
                        Elimina
                    </Button>
                </Modal.Footer>
            </Modal>
        </div>
    );

};

export default CalendarComponent;
-e \n\n
-e \n\n============================\nFile: src/components/Dashboard/
Calendar/Calendar.css\n============================\n
/* src/components/Dashboard/Calendar/Calendar.css */

.calendar-container {
    width: 100%;
    max-width: 1200px;
    margin: 0 auto;
}-e \n\n
-e \n\n============================\nFile: src/components/Dashboard/
Dashboard.css\n============================\n
/* src/components/Dashboard/Dashboard.css */

.dashboard-container {
    display: flex;
    flex-direction: column;
    padding: 20px;
    gap: 20px;
}-e \n\n
-e \n\n============================\nFile: src/components/Dashboard/
Dashboard.jsx\n============================\n
// src/components/Dashboard/Dashboard.jsx

import React from 'react';
import './Dashboard.css';
import Calendar from './Calendar/Calendar';
import Preview from './Preview/Preview';
import { ToastContainer, toast } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import { useNavigate } from 'react-router-dom';
import authService from '../../services/authService';

const Dashboard = ({ setSelectedBoard, boards, setBoards }) => {
    const navigate = useNavigate();
```

```
    const handleLogout = () => {
        authService.logout();
        navigate('/auth/login');
        toast.success('Logout effettuato con successo.');
    };

    return (
        <div className="dashboard-container">
            <button className="logout-button" onClick={handleLogout}
>Logout</button>
            <Calendar />
            <Preview setSelectedBoard={setSelectedBoard}
navigate={navigate} />
            <ToastContainer />
        </div>
    );
};

export default Dashboard;
-e \n\n
-e \n\n==============================\nFile: src/components/Dashboard/
Preview/Preview.css\n==============================\n
/* src/components/Dashboard/Preview/Preview.css */

.preview-container {
    width: 100%;
    max-width: 1200px;
    margin: 0 auto;
}

.card-slide {
    padding: 10px;
}

.card-img-top {
    height: 150px;
    object-fit: cover;
}

.clickable-card {
    cursor: pointer;
    transition: transform 0.2s ease;
}

.clickable-card:hover {
    transform: scale(1.05);
}-e \n\n
-e \n\n==============================\nFile: src/components/Dashboard/
Preview/Preview.jsx\n==============================\n
import React, { useState, useEffect } from 'react';
import Slider from 'react-slick';
import api from '../../../services/api';
import { Card } from 'react-bootstrap';
import './Preview.css';
import { toast } from 'react-toastify';
```

```
import { useNavigate } from 'react-router-dom';

const Preview = ({ setSelectedBoard }) => {
  const [boards, setBoards] = useState([]);
  const navigate = useNavigate();

  const fetchBoards = async () => {
    try {
      const response = await api.get('/auth/boards');

      // Rimuove duplicati in base all'ID della lavagnetta
      const uniqueBoards = response.data.reduce((acc, current) => {
        const x = acc.find(board => board.id === current.id);
        if (!x) return acc.concat([current]);
        else return acc;
      }, []);

      setBoards(uniqueBoards);
    } catch (error) {
      console.error('Errore nel recuperare le lavagnette:', error);
      toast.error('Errore nel recuperare le lavagnette.');
    }
  };

  useEffect(() => {
    fetchBoards();
  }, []);

  const settings = {
    dots: true,
    infinite: false,
    speed: 500,
    slidesToShow: 4,
    slidesToScroll: 1,
    responsive: [
      {
        breakpoint: 1024,
        settings: {
          slidesToShow: 3,
          slidesToScroll: 1,
          infinite: false,
          dots: true,
        },
      },
      {
        breakpoint: 600,
        settings: {
          slidesToShow: 2,
          slidesToScroll: 1,
          initialSlide: 2,
          infinite: false,
        },
      },
      {
        breakpoint: 480,
        settings: {
```

```
          slidesToShow: 1,
          slidesToScroll: 1,
          infinite: false,
        },
      },
    ],
  };

  const defaultBoard = {
    id: 'default',
    name: 'Lavagnetta di Default',
    background: 'default-background.jpg',
  };

  const displayBoards = boards.length > 0 ? boards : [defaultBoard];

  const handleOpenBoard = (board) => {
    setSelectedBoard(board);
    navigate('/board');
  };

  return (
    <div className="preview-container">
      <h3>Anteprima Lavagnette</h3>
      <Slider {...settings}>
        {displayBoards.map((board) => (
          <div key={board.id} className="card-slide">
            <Card className="h-100 clickable-card" onClick={() =>
handleOpenBoard(board)}>
              <Card.Img variant="top" src={`/images/${board.background}
`} alt={board.name} />
              <Card.Body className="d-flex flex-column">
                <Card.Title>{board.name}</Card.Title>
              </Card.Body>
            </Card>
          </div>
        ))}
      </Slider>
    </div>
  );
};

export default Preview;
-e \n\n
-e \n\n==============================\nFile: src/components/
PrivateRoute/PrivateRoute.jsx\n==============================\n
// src/components/PrivateRoute/PrivateRoute.jsx

import React from 'react';
import { Navigate } from 'react-router-dom';
import authService from '../../services/authService';

const PrivateRoute = ({ children }) => {
  const isAuthenticated = authService.isLoggedIn();

  return isAuthenticated ? children : <Navigate to="/auth/login" />;
```

```
};

export default PrivateRoute;
-e \n\n
-e \n\n==============================\nFile: src/components/Header/
Header.jsx\n==============================\n
// src/components/Header/Header.jsx

// src/components/Header/Header.jsx

import React, { useState, useEffect, useRef, useCallback } from 'react';
import './Header.css';
import { toast } from 'react-toastify';
import { useNavigate } from 'react-router-dom';
import authService from '../../services/authService';
import { fetchBoards } from './Actions/fetchBoards';
import { addBoard as addBoardAction } from './Actions/addBoard';
import { deleteBoard as deleteBoardAction } from './Actions/
deleteBoard';
import { updateBoardName as updateBoardNameAction } from './Actions/
updateBoardName';
import { changeBackground as changeBackgroundAction } from './Actions/
changeBackground';

const Header = ({ selectedBoard, setSelectedBoard, boards, setBoards })
=> {
  const [isDropdownOpen, setIsDropdownOpen] = useState(false);
  const [newBoardName, setNewBoardName] = useState('');
  const [selectedBackground, setSelectedBackground] =
useState('blackboard.jpg');
  const [editingBoardId, setEditingBoardId] = useState(null);
  const [editedBoardName, setEditedBoardName] = useState('');
  const editInputRef = useRef(null);
  const navigate = useNavigate();
  const [longPressTimeout, setLongPressTimeout] = useState(null);

  const fetchBoardsCallback = useCallback(() => {
    fetchBoards(setBoards);
  }, [setBoards]);

  useEffect(() => {
    fetchBoardsCallback();
  }, [fetchBoardsCallback]);

  useEffect(() => {
    const initializeSelectedBoard = () => {
      const storedBoard = localStorage.getItem('selectedBoard');
      if (storedBoard) {
        const parsedBoard = JSON.parse(storedBoard);
        const exists = boards.find(board => board.id ===
parsedBoard.id);
        if (exists) {
          setSelectedBoard(exists);
          setSelectedBackground(exists.background || 'blackboard.jpg');
          return;
        }
```

```
      }
      const defaultBoard = boards.find(board => board.is_default === 1);
      if (defaultBoard) {
        setSelectedBoard(defaultBoard);
        setSelectedBackground(defaultBoard.background ||
'blackboard.jpg');
        localStorage.setItem('selectedBoard',
JSON.stringify(defaultBoard));
      } else if (boards.length > 0) {
        setSelectedBoard(boards[0]);
        setSelectedBackground(boards[0].background || 'blackboard.jpg');
        localStorage.setItem('selectedBoard',
JSON.stringify(boards[0]));
      }
    };
    initializeSelectedBoard();
  }, [boards, setSelectedBoard, setSelectedBackground]);

  useEffect(() => {
    if (editingBoardId && editInputRef.current) {
      editInputRef.current.focus();
    }
  }, [editingBoardId]);

  const toggleDropdown = () => {
    setIsDropdownOpen(prev => !prev);
  };

  const selectBoard = (board) => {
    if (!board) {
      toast.error('Errore: lavagnetta non trovata.');
      return;
    }
    setSelectedBoard(board);
    setSelectedBackground(board.background || 'blackboard.jpg');
    setIsDropdownOpen(false);
    toast.success(`Selezionata la lavagnetta "${board.name}".`);
    localStorage.setItem('selectedBoard', JSON.stringify(board));
  };

  const handleAddBoard = async () => {
    if (!newBoardName.trim()) {
      toast.error("Inserisci un nome valido per la lavagnetta.");
      return;
    }
    try {
      await addBoardAction(newBoardName, selectedBackground, boards,
setBoards, setNewBoardName);
      toast.success('Lavagnetta aggiunta con successo.');
    } catch (error) {
      toast.error('Errore nell\'aggiunta della lavagnetta.');
    }
  };

  const handleDeleteSelectedBoard = async () => {
    if (!selectedBoard || selectedBoard.is_default) {
```

```
      toast.error('La lavagnetta di default non può essere eliminata.');
      return;
    }
    try {
      await deleteBoardAction(
        selectedBoard.id,
        boards,
        setBoards,
        selectedBoard,
        setSelectedBoard,
        setSelectedBackground,
        () => {
          navigate('/dashboard');
          toast.success(`Lavagnetta "${selectedBoard.name}"
eliminata.`);
          localStorage.removeItem('selectedBoard');
        }
      );
    } catch (error) {
      toast.error('Errore nell\'eliminazione della lavagnetta.');
    }
  };

  const handleUpdateSelectedBoardName = async () => {
    if (!editedBoardName.trim()) {
      toast.error("Inserisci un nome valido per la lavagnetta.");
      return;
    }
    try {
      await updateBoardNameAction(
        selectedBoard.id,
        editedBoardName,
        selectedBackground,
        boards,
        setBoards,
        selectedBoard,
        setSelectedBoard,
        setEditingBoardId,
        setEditedBoardName
      );
      toast.success(`Nome lavagnetta aggiornato a "$
{editedBoardName}".`);
      const updatedBoard = { ...selectedBoard, name: editedBoardName };
      localStorage.setItem('selectedBoard',
JSON.stringify(updatedBoard));
    } catch (error) {
      toast.error('Errore nell\'aggiornamento del nome della
lavagnetta.');
    }
  };

  const handleChangeBackground = async (background) => {
    if (!selectedBoard) {
      toast.error('Errore: lavagnetta non selezionata.');
      return;
    }
```

```
    try {
      await changeBackgroundAction(
        background,
        selectedBoard,
        setSelectedBackground,
        setSelectedBoard,
        boards,
        setBoards
      );
      toast.info('Sfondo della lavagnetta aggiornato.');
      const updatedBoard = { ...selectedBoard, background };
      localStorage.setItem('selectedBoard',
JSON.stringify(updatedBoard));
    } catch (error) {
      toast.error('Errore nell\'aggiornamento dello sfondo della
lavagnetta.');
    }
  };

  const handleLogout = (e) => {
    e.stopPropagation();
    authService.logout();
    navigate('/auth/login');
    toast.success('Logout effettuato con successo.');
    localStorage.removeItem('selectedBoard');
  };

  const handleHomeClick = (e) => {
    e.stopPropagation();
    navigate('/dashboard');
    toast.success('Sei tornato alla dashboard.');
  };

  const handleMouseDownOnTitle = () => {
    if (!selectedBoard) return;
    setLongPressTimeout(
      setTimeout(() => {
        setEditingBoardId(selectedBoard.id);
        setEditedBoardName(selectedBoard.name);
      }, 500)
    );
  };

  const handleMouseUpOnTitle = () => {
    if (longPressTimeout) {
      clearTimeout(longPressTimeout);
      setLongPressTimeout(null);
    }
  };

  return (
    <header className="header-container">
      <div className="header-top">
        <h1
          onDoubleClick={handleDeleteSelectedBoard}
          onMouseDown={handleMouseDownOnTitle}
```

```
          onMouseUp={handleMouseUpOnTitle}
          aria-label="Titolo lavagnetta selezionata"
          onClick={(e) => {
            e.stopPropagation();
            toggleDropdown();
          }}
        >
          {editingBoardId !== selectedBoard?.id ? (
            selectedBoard ? (
              selectedBoard.name
            ) : (
              'Nessuna Lavagnetta Selezionata'
            )
          ) : (
            <input
              ref={editInputRef}
              value={editedBoardName}
              onChange={(e) => setEditedBoardName(e.target.value)}
              onBlur={handleUpdateSelectedBoardName}
              onKeyDown={(e) => {
                if (e.key === 'Enter') {
                  handleUpdateSelectedBoardName();
                } else if (e.key === 'Escape') {
                  setEditingBoardId(null);
                  setEditedBoardName('');
                }
              }}
              placeholder="Modifica Lavagnetta"
              aria-label="Modifica Lavagnetta"
            />
          )}
        </h1>
        <div className="header-buttons">
          <button className="logout-button" onClick={handleLogout} aria-
label="Logout">
            Logout
          </button>
          <button className="home-button" onClick={handleHomeClick}
aria-label="Home">
            Home
          </button>
        </div>
      </div>
      {isDropdownOpen && (
        <div className="header-bottom">
          <div className="boards-row">
            {boards.map((board) => (
              <span
                key={board.id}
                className={`board-item ${selectedBoard?.id ===
board.id ? 'selected' : ''} ${board.is_default ? 'default-board' : ''}`}
                onClick={() => selectBoard(board)}
                role="button"
                tabIndex={0}
                aria-label={`Seleziona lavagnetta ${board.name}`}
              >
```

```
                {board.name}
              </span>
            ))}
          </div>
          <div className="add-board">
            <input
              type="text"
              value={newBoardName}
              onChange={(e) => setNewBoardName(e.target.value)}
              placeholder="Aggiungi Lavagnetta"
              onKeyDown={(e) => {
                if (e.key === 'Enter') {
                  handleAddBoard();
                }
              }}
              aria-label="Aggiungi Lavagnetta"
            />
            <button onClick={handleAddBoard} aria-label="Aggiungi
Lavagnetta">
              Aggiungi
            </button>
          </div>
          <div className="background-selector">
            <label htmlFor="background-choice">Scegli lo sfondo della
lavagnetta </label>
            <select
              id="background-choice"
              value={selectedBackground}
              onChange={(e) => handleChangeBackground(e.target.value)}
              aria-label="Scegli lo sfondo della lavagnetta"
            >
              <option value="blackboard.jpg">Blackboard</option>
              <option value="frettolosa.jpg">Frettolosa</option>
              <option value="gesso.jpg">Gesso</option>
              <option value="lavata.jpg">Lavata</option>
              <option value="nera.jpg">Nera</option>
              <option value="graffiata.jpg">Graffiata</option>
              <option value="verde.jpg">Verde</option>
            </select>
          </div>
        </div>
      )}
      {!isDropdownOpen && (
        <div
          className="arrow-toggle"
          onClick={toggleDropdown}
          role="button"
          tabIndex={0}
          aria-label="Mostra/Nascondi Lavagnette"
        >
          <div className="arrow"></div>
        </div>
      )}
    </header>
  );
};
```

```
export default Header;
-e \n\n
-e \n\n============================\nFile: src/components/Header/
Actions/deleteBoard.js\n============================\n
// src/components/Header/actions/deleteBoard.js

import api from '../../../services/api';
import { toast } from 'react-toastify';

export const deleteBoard = async (
    boardId,
    boards,
    setBoards,
    selectedBoard,
    setSelectedBoard,
    setSelectedBackground,
    onSuccess // Aggiunto parametro di callback
) => {
    const boardToDelete = boards.find((board) => board.id === boardId);
    if (!boardToDelete) return;

    // Mostra il messaggio di conferma prima di procedere
    const userConfirmed = window.confirm(`Sei sicuro di voler eliminare
la lavagnetta "${boardToDelete.name}"?`);
    if (!userConfirmed) return;

    if (boardToDelete.is_default) {
        toast.error('La lavagnetta di default non può essere
eliminata.');
        return;
    }

    try {
        await api.delete(`/auth/boards/${boardId}`);
        setBoards((prevBoards) => prevBoards.filter((board) =>
board.id !== boardId));

        // Verifica se la lavagnetta selezionata è stata eliminata
        if (selectedBoard && selectedBoard.id === boardId) {
            setSelectedBoard(null);
            setSelectedBackground('blackboard.jpg');
            toast.error('Nessuna lavagnetta selezionata.');
        }

        toast.success(`Lavagnetta "${boardToDelete.name}" eliminata con
successo.`);

        // Verifica la funzione di callback onSuccess solo se ancora
definita
        if (typeof onSuccess === 'function') onSuccess();
    } catch (error) {
        console.error("Errore nell'eliminazione della lavagnetta:",
error);
        toast.error("Errore nell'eliminazione della lavagnetta. Riprova
più tardi.");
```

```
    }
};
-e \n\n
-e \n\n============================\nFile: src/components/Header/
Actions/changeBackground.js\n============================\n
// src/components/Header/actions/changeBackground.js

import api from '../../../services/api';
import { toast } from 'react-toastify';

export const changeBackground = async (
    background,
    selectedBoard,
    setSelectedBackground,
    setSelectedBoard,
    boards,
    setBoards
) => {
    if (!selectedBoard) return;
    try {
        const response = await api.put(`/auth/boards/${selectedBoard.id}
`, {
            name: selectedBoard.name,
            background,
        });
        setSelectedBackground(response.data.background);
        setSelectedBoard({ ...selectedBoard, background:
response.data.background });
        setBoards(
            boards.map((board) =>
                board.id === selectedBoard.id ? { ...board, background:
response.data.background } : board
            )
        );
        toast.success('Sfondo della lavagnetta aggiornato con
successo.');
    } catch (error) {
        console.error('Errore nell\'aggiornamento dello sfondo:',
error);
        toast.error('Errore nell\'aggiornamento dello sfondo. Riprova
più tardi.');
    }
};
-e \n\n
-e \n\n============================\nFile: src/components/Header/
Actions/fetchBoards.js\n============================\n
// src/components/Header/actions/fetchBoards.js

import api from '../../../services/api';
import { toast } from 'react-toastify';

export const fetchBoards = async (setBoards) => {
    try {
        const response = await api.get('/auth/boards');
        setBoards(response.data || []);
    } catch (error) {
```

```
        console.error('Errore nel recuperare le lavagnette:', error);
        toast.error('Errore nel recuperare le lavagnette.');
    }
};
-e \n\n
-e \n\n============================\nFile: src/components/Header/
Actions/updateBoardName.js\n============================\n
// src/components/Header/actions/updateBoardName.js

import api from '../../../services/api';
import { toast } from 'react-toastify';

export const updateBoardName = async (
    boardId,
    editedBoardName,
    selectedBackground,
    boards,
    setBoards,
    selectedBoard,
    setSelectedBoard,
    setEditingBoardId,
    setEditedBoardName
) => {
    if (editedBoardName.trim() === '') {
        toast.error('Il nome della lavagnetta non può essere vuoto.');
        return;
    }
    try {
        const response = await api.put(`/auth/boards/${boardId}`, {
            name: editedBoardName,
            background: selectedBackground,
        });
        setBoards(
            boards.map((board) =>
                board.id === boardId ? { ...board, name:
response.data.name } : board
            )
        );
        if (selectedBoard && selectedBoard.id === boardId) {
            setSelectedBoard({ ...selectedBoard, name:
response.data.name });
        }
        setEditingBoardId(null);
        setEditedBoardName('');
        toast.success('Nome della lavagnetta aggiornato con successo.');
    } catch (error) {
        console.error('Errore nell\'aggiornamento della lavagnetta:',
error);
        toast.error('Errore nell\'aggiornamento della lavagnetta.
Riprova più tardi.');
    }
};
-e \n\n
-e \n\n============================\nFile: src/components/Header/
Actions/addBoard.js\n============================\n
// src/components/Header/actions/addBoard.js
```

```
import api from '../../../services/api';
import { toast } from 'react-toastify';

export const addBoard = async (
    newBoardName,
    selectedBackground,
    boards,
    setBoards,
    setNewBoardName
) => {
    if (newBoardName.trim() === '') {
        toast.error('Il nome della lavagnetta non può essere vuoto.');
        return;
    }
    try {
        const response = await api.post(`/auth/boards`, {
            name: newBoardName,
            background: selectedBackground,
        });
        setBoards([...boards, response.data]);
        setNewBoardName('');
        toast.success('Lavagnetta aggiunta con successo.');
    } catch (error) {
        console.error('Errore nella creazione della lavagnetta:',
error);
        toast.error('Errore nella creazione della lavagnetta. Riprova
più tardi.');
    }
};
-e \n\n
-e \n\n==============================\nFile: src/components/Header/
Header.css\n==============================\n
/* src/components/Header/Header.css */
@import url('https://fonts.googleapis.com/css2?
family=Rock+Salt&display=swap');

/* Variabili di stile per React */
:root {
    --header-height: 60px;
    --transition-duration: 0.3s;
    --arrow-size: 16px;
}

/* Contenitore principale dell'header */
.header-container {
    position: fixed;
    top: 0;
    left: 0;
    width: 100%;
    height: var(--header-height);
    display: flex;
    flex-direction: column;
    padding: 0 20px;
    /* Rimosso padding verticale per evitare sovrapposizioni */
    color: #fff;
```

```
    box-sizing: border-box;
    z-index: 1010;
    background-color: transparent;
    /* Rimosso lo sfondo grigio */
    transition: transform var(--transition-duration) ease, opacity
var(--transition-duration) ease;
}

/* Nasconde l'header con transizione */
.header-container.header-hidden {
    transform: translateY(-100%);
    opacity: 0;
    pointer-events: none;
}

/* Parte superiore dell'header */
.header-top {
    display: flex;
    justify-content: space-between;
    align-items: center;
    width: 100%;
    cursor: pointer;
    padding: 5px 10px;
    height: 100%;
    transition: opacity var(--transition-duration) ease, transform
var(--transition-duration) ease;
}

/* Titolo dell'header */
.header-top h1 {
    font-family: "Rock Salt", cursive;
    margin: 0;
    font-size: 1.5em;
    color: #ffaa33;
    letter-spacing: 2px;
}

/* Input per la modifica del titolo */
.header-top input {
    font-family: "Rock Salt", cursive;
    font-size: 1.5em;
    color: #ffaa33;
    background-color: transparent;
    border: none;
    outline: none;
    width: 100%;
}

/* Stile per i pulsanti Logout e Home */
.header-buttons {
    display: flex;
    flex-direction: column;
    align-items: flex-end;
}

/* Stile per il pulsante Logout */
```

```
.logout-button {
    background-color: transparent;
    border: none;
    color: #fff;
    font-family: 'Rock Salt', cursive;
    font-size: 1em;
    cursor: pointer;
    transition: color var(--transition-duration) ease, transform 0.2s
ease;
    padding: 8px 16px;
    margin-bottom: 5px;
}

.logout-button:hover {
    color: #ff8c00;
    transform: scale(1.1);
}

.logout-button:active {
    transform: scale(0.95);
}

/* Stile per il pulsante Home */
.home-button {
    font-family: 'Rock Salt', cursive;
    font-size: 1em;
    color: #fff;
    background-color: #007bff;
    /* Colore di sfondo blu */
    border: none;
    cursor: pointer;
    padding: 8px 16px;
    border-radius: 4px;
    transition: background-color 0.3s ease, transform 0.2s ease;
}

.home-button:hover {
    background-color: #0056b3;
    /* Colore più scuro al passaggio del mouse */
    transform: scale(1.05);
}

.home-button:active {
    transform: scale(0.95);
}

/* Parte inferiore dell'header */
.header-bottom {
    position: absolute;
    top: 100%;
    left: 0;
    right: 0;
    background-color: #222;
    padding: 20px;
    box-sizing: border-box;
    z-index: 1011;
```

```
    transition: max-height var(--transition-duration) ease-out, opacity
var(--transition-duration) ease-out;
    opacity: 1;
    visibility: visible;
    max-height: 100vh;
    overflow: hidden;
}

.header-bottom.show {
    max-height: 100vh;
    opacity: 1;
    visibility: visible;
}

/* Nasconde la parte dell'header */
.header-container.hidden {
    opacity: 0;
    transform: translateY(-100%);
}

/* Stile per il bottone comune */
.button-common {
    background-color: transparent !important;
    border: none !important;
    color: inherit !important;
    font-family: "Rock Salt", cursive !important;
    font-size: 0.6em;
    cursor: pointer;
    transition: color var(--transition-duration) ease, transform 0.2s
ease;
}

.button-common:hover {
    color: #ff8c00;
    transform: scale(1.2);
}

.button-common:active {
    transform: scale(0.95);
}

/* Stili per bottoni specifici dell'header */
.header-bottom button,
.header-bottom .btn-outline-warning,
.add-board button,
.choose-background {
    background-color: transparent !important;
    border: none !important;
    color: inherit !important;
    font-family: "Rock Salt", cursive !important;
    font-size: 0.6em;
    cursor: pointer;
    transition: color var(--transition-duration) ease, transform 0.2s
ease;
}
```

```
.header-bottom button:hover,
.header-bottom .btn-outline-warning:hover,
.add-board button:hover,
.choose-background:hover {
    color: #ff8c00;
    transform: scale(1.2);
}

.header-bottom button:active,
.header-bottom .btn-outline-warning:active,
.add-board button:active,
.choose-background:active {
    transform: scale(0.95);
}

/* Contenitore per la freccia per mostrare/nascondere l'header */
.arrow-toggle {
    position: fixed;
    top: 10px;
    left: 50%;
    transform: translateX(-50%);
    z-index: 1012;
    cursor: pointer;
    transition: transform var(--transition-duration) ease;
}

.arrow-toggle .arrow {
    width: var(--arrow-size);
    height: var(--arrow-size);
    background-color: #ffaa33;
    clip-path: polygon(50% 100%, 0% 0%, 100% 0%);
    transition: background-color var(--transition-duration) ease,
transform var(--transition-duration) ease;
}

.arrow-toggle:hover .arrow {
    background-color: #ffaa33;
    transform: scale(1.1);
}

.arrow-toggle:active .arrow {
    transform: scale(0.9);
}

/* Contenitore per l'aggiunta di una nuova lavagnetta */
.add-board {
    display: flex;
    align-items: center;
    border: none !important;
}

/* Input per il nome della lavagnetta */
.add-board input {
    padding: 5px;
    border: none !important;
    border-radius: 3px 0 0 3px;
```

```
    font-family: "Rock Salt", cursive;
    background-color: transparent;
    color: #ffaa33;
    transition: border-color var(--transition-duration) ease;
    font-size: 1em;
}

.add-board input:focus {
    outline: none;
    border-bottom: 1px solid #ffaa33;
}

/* Contenitore per le righe delle lavagnette */
.boards-row {
    display: flex;
    align-items: center;
    overflow-x: auto;
    white-space: nowrap;
    padding-bottom: 5px;
    scrollbar-width: none;
    font-family: "Rock Salt", cursive;
    font-size: 1.5em;
}

.boards-row::-webkit-scrollbar {
    display: none;
}

/* Stile degli elementi della lavagnetta */
.board-item {
    margin-right: 15px;
    color: #fff;
    font-size: 1em;
    cursor: pointer;
    transition: color var(--transition-duration) ease, font-size 0.3s
ease;
}

.board-item.selected {
    color: #ffaa33;
    font-size: 1.2em;
}

.board-item:hover {
    color: #ff8c00;
}

/* Bottone per la scelta dello sfondo */
#background-choice {
    background-color: transparent !important;
    border: none !important;
    border-bottom: 1px solid #fff;
    font-family: "Rock Salt", cursive !important;
    font-size: 1.2em !important;
    transition: color var(--transition-duration) ease, transform 0.2s
ease;
```

```
    color: #39ff14 !important;
}

/* Bottone per la chat */
.chat-button {
    background-color: #25d366;
    border: none;
    border-radius: 50%;
    color: white;
    width: 40px;
    height: 40px;
    font-size: 1.5em;
    cursor: pointer;
    position: fixed;
    bottom: 20px;
    right: 20px;
    z-index: 1001;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
    transition: background-color 0.3s;
}

.chat-button:hover {
    background-color: #1da851;
}-e \n\n
-e \n\n==============================\nFile: src/
App.test.js\n==============================\n
import { render, screen } from '@testing-library/react';
import App from './App';

test('renders learn react link', () => {
  render(<App />);
  const linkElement = screen.getByText(/learn react/i);
  expect(linkElement).toBeInTheDocument();
});
-e \n\n
-e \n\n==============================\nFile: src/
App.jsx\n==============================\n
// src/App.jsx

import React, { useState, useEffect, useRef } from 'react';
import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-
router-dom';
import Login from './components/Auth/Login/Login';
import Register from './components/Auth/Register/Register';
import Dashboard from './components/Dashboard/Dashboard';
import Board from './components/Board/Board';
import Header from './components/Header/Header';
import './App.css';
import { ToastContainer } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import authService from './services/authService';
import jwt_decode from 'jwt-decode';

const App = () => {
    const [authenticated, setAuthenticated] =
useState(authService.isLoggedIn());
```

```
    const [selectedBoard, setSelectedBoard] = useState(null);
    const [boards, setBoards] = useState([]);

    const inactivityTimeoutRef = useRef(null);
    const tokenExpiryTimeoutRef = useRef(null);
    const lastActivityRef = useRef(Date.now());

    useEffect(() => {
        const handleLogout = () => {
            authService.logout();
            setAuthenticated(false); // Aggiorna lo stato di
autenticazione
        };

        const resetTimers = () => {
            clearTimeout(inactivityTimeoutRef.current);
            clearTimeout(tokenExpiryTimeoutRef.current);
            startInactivityTimer();
            startTokenExpiryTimer();
        };

        const handleInactivity = () => {
            console.log('Inattività rilevata, eseguo logout e
reindirizzo al login.');
            handleLogout();
        };

        const startInactivityTimer = () => {
            inactivityTimeoutRef.current = setTimeout(handleInactivity,
10000); // 10 secondi
        };

        const handleTokenExpiry = async () => {
            const token = authService.getToken();
            if (!token) {
                handleLogout();
                return;
            }

            try {
                const decoded = jwt_decode(token);
                const currentTime = Date.now() / 1000;
                const timeRemaining = decoded.exp - currentTime;

                if (timeRemaining <= 0) {
                    handleLogout();
                } else if (timeRemaining <= 10) {
                    const inactivityDuration = (Date.now() -
lastActivityRef.current) / 1000;
                    if (inactivityDuration > 10) {
                        handleLogout();
                    } else {
                        try {
                            await authService.renewToken();
                            console.log('Token rinnovato con
successo.');
```

```
                            resetTimers();
                        } catch (error) {
                            console.error('Errore durante il rinnovo del
token:', error);
                            handleLogout();
                        }
                    }
                }
            } catch (error) {
                console.error('Errore nel decodificare il token:',
error);
                handleLogout();
            }
        };

        const startTokenExpiryTimer = () => {
            const token = authService.getToken();
            if (!token) return;

            try {
                const decoded = jwt_decode(token);
                const currentTime = Date.now() / 1000;
                const timeRemaining = decoded.exp - currentTime;
                const triggerTime = (timeRemaining - 10) * 1000;

                if (triggerTime > 0) {
                    tokenExpiryTimeoutRef.current =
setTimeout(handleTokenExpiry, triggerTime);
                } else {
                    handleTokenExpiry();
                }
            } catch (error) {
                console.error('Errore nel decodificare il token:',
error);
                handleLogout();
            }
        };

        const updateLastActivity = () => {
            lastActivityRef.current = Date.now();
            resetTimers();
        };

        const events = ['mousemove', 'click', 'keydown', 'scroll'];
        events.forEach((event) => {
            window.addEventListener(event, updateLastActivity);
        });

        resetTimers();

        return () => {
            events.forEach((event) => {
                window.removeEventListener(event, updateLastActivity);
            });
            clearTimeout(inactivityTimeoutRef.current);
            clearTimeout(tokenExpiryTimeoutRef.current);
```

```
        };
    }, []);

    return (
        <Router>
            <div>
                <ToastContainer />
                <Routes>
                    <Route path="/auth/login" element={<Login
setAuthenticated={setAuthenticated} />} />
                    <Route path="/auth/register" element={<Register /
>} />
                    <Route
                        path="/dashboard"
                        element={authenticated ? <Dashboard
setSelectedBoard={setSelectedBoard} boards={boards}
setBoards={setBoards} /> : <Navigate to="/auth/login" />}
                    />
                    <Route
                        path="/board"
                        element={
                            authenticated ? (
                                <>
                                    <Header
selectedBoard={selectedBoard} setSelectedBoard={setSelectedBoard}
boards={boards} setBoards={setBoards} />
                                    <Board selectedBoard={selectedBoard}
/>
                                </>
                            ) : (
                                <Navigate to="/auth/login" />
                            )
                        }
                    />
                    <Route path="/" element={<Navigate to="/dashboard" /
>} />
                </Routes>
            </div>
        </Router>
    );
};

export default App;
-e \n\n
-e \n\n==============================\nFile: src/
setupTests.js\n==============================\n
// jest-dom adds custom jest matchers for asserting on DOM nodes.
// allows you to do things like:
// expect(element).toHaveTextContent(/react/i)
// learn more: https://github.com/testing-library/jest-dom
import '@testing-library/jest-dom';
-e \n\n
-e \n\n==============================\nFile: src/services/
boardService.js\n==============================\n
import api from './api';
import axios from 'axios';
```

```
/**
 * Ottiene tutte le lavagnette dell'utente.
 * @returns {Promise<Array>} - Array di lavagnette.
 */
const getBoards = async () => {
    try {
        const response = await api.get('/boards');
        return response.data;
    } catch (error) {
        throw error;
    }
};

/**
 * Crea una nuova lavagnetta.
 * @param {string} name - Nome della lavagnetta.
 * @param {string} background - Sfondo della lavagnetta.
 * @returns {Promise<Object>} - Lavagnetta creata.
 */
const createBoard = async (name, background) => {
    try {
        const response = await api.post('/boards', { name,
background });
        return response.data;
    } catch (error) {
        throw error;
    }
};

/**
 * Aggiorna una lavagnetta.
 * @param {number} id - ID della lavagnetta.
 * @param {string} name - Nuovo nome.
 * @param {string} background - Nuovo sfondo.
 * @returns {Promise<Object>} - Lavagnetta aggiornata.
 */
const updateBoard = async (id, name, background) => {
    try {
        const response = await api.put(`/boards/${id}`, { name,
background });
        return response.data;
    } catch (error) {
        throw error;
    }
};

/**
 * Elimina una lavagnetta.
 * @param {number} id - ID della lavagnetta.
 * @returns {Promise<Object>} - Messaggio di conferma.
 */
const deleteBoard = async (id) => {
    try {
        const response = await api.delete(`/boards/${id}`);
```

```
        return response.data;
    } catch (error) {
        throw error;
    }
};

/**
 * Aggiorna il nome di una lavagnetta.
 * @param {number} id - ID della lavagnetta.
 * @param {string} name - Nuovo nome.
 * @returns {Promise<Object>} - Lavagnetta aggiornata.
 */
const updateBoardName = async (id, name) => {
    try {
        const response = await api.put(`/boards/${id}`, { name });
        return response.data;
    } catch (error) {
        throw error;
    }
};

/**
 * Aggiorna lo sfondo di una lavagnetta.
 * @param {number} id - ID della lavagnetta.
 * @param {string} background - Nuovo sfondo.
 * @returns {Promise<Object>} - Lavagnetta aggiornata.
 */
const updateBoardBackground = async (id, background) => {
    try {
        const response = await api.put(`/boards/${id}`, { background });
        return response.data;
    } catch (error) {
        throw error;
    }
};

const boardService = {
    getBoards,
    createBoard,
    updateBoard,
    deleteBoard,
    updateBoardName,
    updateBoardBackground
};

export default boardService;
-e \n\n
-e \n\n==============================\nFile: src/services/
noteService.js\n==============================\n
// src/services/noteService.js

import api from './api';

/**
 * Recupera tutte le note dell'utente.
 * @returns {Promise<Array>} - Array di note.
```

```
 */
export const getNotes = async () => {
    try {
        const response = await api.get('/auth/notes');
        return response.data;
    } catch (error) {
        console.error('Errore nel recuperare le note:', error);
        throw error;
    }
};
-e \n\n
-e \n\n============================\nFile: src/services/
authService.js\n============================\n
// src/services/authService.js

import axios from 'axios';
import jwt_decode from 'jwt-decode';

const API_URL = 'http://localhost:3000/api/auth'; // Assicurati che
questo sia corretto

/**
 * Registra un nuovo utente e salva il token.
 * @param {string} email - Email dell'utente.
 * @param {string} password - Password dell'utente.
 * @param {string} username - Username dell'utente.
 * @returns {Object} - Risultato della registrazione.
 */
const register = async (email, password, username) => {
    try {
        const response = await axios.post(`${API_URL}/register`,
{ email, password, username });
        if (response.data.token) {
            setToken(response.data.token);
        }
        return response.data;
    } catch (error) {
        console.error("Errore durante la registrazione:", error);
        throw error;
    }
};

/**
 * Effettua il login dell'utente e salva il token.
 * @param {string} email - Email dell'utente.
 * @param {string} password - Password dell'utente.
 * @returns {Object} - Risultato del login.
 */
const login = async (email, password) => {
    try {
        const response = await axios.post(`${API_URL}/login`, { email,
password });
        if (response.data.token) {
            setToken(response.data.token);
        }
        return response.data;
```

```
    } catch (error) {
        console.error("Errore durante il login:", error);
        throw error;
    }
};

/**
 * Richiede un nuovo token al backend.
 * @returns {Object} - Risultato del rinnovo del token.
 */
const renewToken = async () => {
    try {
        const token = getToken();
        const response = await axios.post(`${API_URL}/renew-token`, {},
{
            headers: {
                'Authorization': `Bearer ${token}`
            }
        });
        if (response.data.token) {
            setToken(response.data.token);
        }
        return response.data;
    } catch (error) {
        console.error("Errore durante il rinnovo del token:", error);
        throw error;
    }
};

/**
 * Resetta completamente la sessione dell'utente.
 */
const resetSession = () => {
    localStorage.removeItem('token');
    localStorage.removeItem('lastActivity');
};

/**
 * Effettua il logout dell'utente, distruggendo la sessione e
reindirizzando al login.
 */
const logout = () => {
    resetSession();
    window.location.replace('/auth/login'); // Reindirizza alla pagina
di login
};

/**
 * Salva il token nel localStorage.
 * @param {string} token - Token JWT.
 */
const setToken = (token) => {
    localStorage.setItem('token', token);
    localStorage.setItem('lastActivity', Date.now());
    console.log('Token salvato nel localStorage:', token);
};
```

```
/**
 * Recupera il token dal localStorage.
 * @returns {string|null} - Token JWT o null se non presente.
 */
const getToken = () => {
    return localStorage.getItem('token');
};

/**
 * Verifica se l'utente è loggato controllando la validità del token.
 * @returns {boolean} - True se loggato e token valido, false
altrimenti.
 */
const isLoggedIn = () => {
    const token = getToken();
    if (!token) {
        console.log('Nessun token trovato.');
        return false;
    }

    try {
        const decoded = jwt_decode(token);
        const currentTime = Date.now() / 1000; // Tempo corrente in
secondi
        const isValid = decoded.exp > currentTime;
        console.log('Token decodificato:', decoded);
        console.log('Token valido:', isValid);
        return isValid;
    } catch (error) {
        console.error("Token non valido:", error);
        return false;
    }
};

const authService = {
    register,
    login,
    renewToken,
    logout,
    setToken,
    getToken,
    isLoggedIn,
    resetSession,
};

export default authService;
-e \n\n
-e \n\n==============================\nFile: src/services/
api.js\n==============================\n
// frontend/src/services/api.js

import axios from 'axios';
import authService from './authService';
```

```
const API_URL = process.env.REACT_APP_API_URL || 'http://localhost:3000/
api';
console.log('API_URL:', API_URL);

const api = axios.create({
    baseURL: API_URL,
});

// Funzione per impostare o aggiornare il timestamp dell'ultima attività
const updateLastActivity = () => {
    const currentTime = Date.now();
    localStorage.setItem('lastActivity', currentTime);
    console.log('Timestamp attività aggiornato:', currentTime);
};

// Funzione per controllare l’inattività con limite di 1 minuto
const checkInactivity = () => {
    const lastActivity = localStorage.getItem('lastActivity');
    const INACTIVITY_LIMIT = 60 * 1000; // 1 minuto in millisecondi

    if (lastActivity) {
        const currentTime = Date.now();
        const timeSinceLastActivity = currentTime - lastActivity;

        if (timeSinceLastActivity > INACTIVITY_LIMIT) {
            console.log('Inattività rilevata, eseguo logout e
reindirizzo al login.');
            authService.logout();
        } else {
            updateLastActivity();
        }
    }
};

// Aggiungi l'interceptor per includere il token in tutte le richieste
api.interceptors.request.use(
    (config) => {
        const token = authService.getToken();
        if (token) {
            config.headers['Authorization'] = `Bearer ${token}`;
            console.log('Token aggiunto alla richiesta:', token);
        } else {
            console.log('Token non trovato, reindirizzo al login.');
        }
        checkInactivity(); // Verifica l'inattività per ogni richiesta
        return config;
    },
    (error) => {
        console.error("Errore nell'interceptor della richiesta:",
error);
        return Promise.reject(error);
    }
);

export default api;
-e \n\n
```

```
alessandrotornabene@4f34e342-5886-4f1a-ba7d-072eedb502eb lavagnetta %
```