JWT Authentication Integration Guide for Spell Check

Overview

This guide shows how to secure your LanguageTool spell check integration with JWT authentication. The solution includes:

1. **Frontend Changes**: Updated SpellCheckPlugin with JWT authentication
2. **Backend API**: Secure proxy endpoint for LanguageTool
3. **Authentication System**: Complete auth context and token management

Setup Instructions

1. Backend Setup

Environment Variables

```bash
# .env file
JWT_SECRET=your-super-secret-jwt-key-here
REFRESH_TOKEN_SECRET=your-refresh-token-secret
LANGUAGETOOL_URL=http://localhost:8010/v2/check
NODE_ENV=production
```

Install Dependencies

```bash
npm install express jsonwebtoken node-fetch express-rate-limit
# or
yarn add express jsonwebtoken node-fetch express-rate-limit
```

Add the Backend Route

1. Copy the backend code from the second artifact
2. Add it to your Express app or API routes
3. Configure CORS if needed:

```javascript
```

```
// CORS configuration for spell check
app.use('/api/spellcheck', cors({
  origin: process.env.FRONTEND_URL || 'http://localhost:3000',
  credentials: true
}));
```

2. Frontend Integration

Install/Update Dependencies

```bash
npm install prop-types
# Ensure you have these if not already installed
npm install @lexical/react lexical @lexical/utils
```

Replace SpellCheckPlugin

1. Replace your existing `SpellCheckPlugin.jsx` with the updated version (first artifact)
2. The new plugin includes:
   - JWT token management
   - Automatic token refresh
   - Error handling for auth failures
   - Graceful fallbacks

Add Authentication Context (Optional)

If you don't have an existing auth system, you can use the provided `AuthContext.js`:

```javascript
// In your App.js or root component
import { AuthProvider } from './path/to/AuthContext';

function App() {
  return (
    <AuthProvider>
      {/* Your app components */}
    </AuthProvider>
  );
}
```

3. Configuration Options

Token Storage

The plugin supports multiple token storage methods. Choose the one that fits your app:

```javascript
// Option 1: localStorage (default)
const token = localStorage.getItem('authToken');

// Option 2: sessionStorage
const token = sessionStorage.getItem('authToken');

// Option 3: React Context (recommended)
const { token } = useAuth();

// Option 4: Cookies
const token = document.cookie.split('; ')
  .find(row => row.startsWith('authToken='))
  ?.split('=')[1];
```

Customizing Authentication URLs

Update the URLs in `LanguageToolService` to match your backend:

```javascript
class LanguageToolService {
  constructor() {
    this.apiUrl = '/api/spellcheck'; // Your backend endpoint
    // ... rest of the constructor
  }

  async refreshTokenIfNeeded() {
    // Update refresh endpoint
    const response = await fetch('/api/auth/refresh', {
      // ... your refresh logic
    });
  }
}
```

4. Integration with Existing Auth System

If you already have an authentication system, modify these parts:

Token Retrieval

```javascript
// In LanguageToolService.getAuthToken()
getAuthToken() {
  // Replace with your auth system's method
  return yourAuthSystem.getToken();
}
```

Token Refresh

```javascript
// In LanguageToolService.refreshTokenIfNeeded()
async refreshTokenIfNeeded() {
  const currentToken = this.getAuthToken();

  if (!this.isTokenValid(currentToken)) {
    // Use your auth system's refresh method
    const newToken = await yourAuthSystem.refreshToken();
    return newToken;
  }

  return currentToken;
}
```

Authentication Failure Handling

```javascript
// In LanguageToolService.handleAuthFailure()
handleAuthFailure() {
  // Use your auth system's logout method
  yourAuthSystem.logout();
  // Or redirect to login
  // window.location.href = '/login';
}
```

5. Security Considerations

Rate Limiting

The backend includes rate limiting (100 requests per 15 minutes per user). Adjust as needed:

```javascript
const spellCheckLimiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 50, // Adjust based on your needs
  // ...
});
```

Text Length Limits

Prevent abuse with text length limits:

```javascript
// In backend spell check endpoint
if (text.length > 10000) { // Adjust limit as needed
  return res.status(400).json({
    error: 'Text too long. Maximum 10,000 characters allowed.',
    code: 'TEXT_TOO_LONG'
  });
}
```

HTTPS in Production

Ensure all requests use HTTPS in production:

```javascript
// Add to your backend
if (process.env.NODE_ENV === 'production' && !req.secure) {
  return res.redirect(301, `https://${req.headers.host}${req.url}`);
}
```

6. Testing the Integration

Test Authentication

```javascript
```

```javascript
// Test the spell check with authentication
const testSpellCheck = async () => {
  try {
    const response = await fetch('/api/spellcheck', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${yourToken}`,
      },
      body: JSON.stringify({
        text: 'This is a tets message.', // intentional typo
        language: 'en-US'
      }),
    });

    const data = await response.json();
    console.log('Spell check results:', data);
  } catch (error) {
    console.error('Spell check failed:', error);
  }
};
```

Test Token Refresh

```javascript
// Test automatic token refresh
const testTokenRefresh = async () => {
  // Use an expired or invalid token
  localStorage.setItem('authToken', 'expired.token.here');

  // Try spell check - should automatically refresh
  // Check network tab to see refresh request
};
```

7. Error Handling

The integration includes comprehensive error handling:

- **401 Unauthorized**: Token expired or invalid → Automatic refresh attempted
- **403 Forbidden**: Insufficient permissions → User notified
- **429 Too Many Requests**: Rate limit exceeded → User advised to wait

- **502 Bad Gateway**: LanguageTool unavailable → Service unavailable message
- **Network Errors**: Connection issues → Graceful fallback

8. Monitoring and Logging

Add logging for monitoring:

```javascript
// In your backend
const winston = require('winston');

const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  transports: [
    new winston.transports.File({ filename: 'spellcheck.log' }),
  ],
});

// Log spell check requests
router.post('/spellcheck', (req, res, next) => {
  logger.info('Spell check request', {
    userId: req.user.userId,
    textLength: req.body.text?.length,
    timestamp: new Date().toISOString(),
  });
  next();
});
```

9. Performance Optimization

Caching

The `LanguageToolService` includes caching. You can enhance it:

```javascript
```

```
// Enhanced caching with expiration
class LanguageToolService {
  constructor() {
    this.cache = new Map();
    this.cacheExpiration = 5 * 60 * 1000; // 5 minutes
  }

  async checkText(text) {
    const cacheKey = text.trim().toLowerCase();
    const cachedResult = this.cache.get(cacheKey);

    if (cachedResult && Date.now() - cachedResult.timestamp < this.cacheExpiration) {
      return cachedResult.data;
    }

    // ... rest of the method

    // Cache with timestamp
    this.cache.set(cacheKey, {
      data: result,
      timestamp: Date.now()
    });

    return result;
  }
}
```

Debouncing

The plugin includes debouncing (1 second). Adjust as needed:

```
javascript

// In SpellCheckPlugin useEffect
timeoutId = setTimeout(async () => {
  // Spell check logic
}, 2000); // Increase to 2 seconds for less frequent checks
```

Troubleshooting

Common Issues

1. **CORS Errors**: Ensure your backend allows requests from your frontend domain

2. **Token Not Found**: Check your token storage method matches the retrieval method
3. **Rate Limiting**: Increase limits or implement user-specific quotas
4. **LanguageTool Connection**: Verify LanguageTool server is running and accessible

Debug Mode

Enable debug logging:

```javascript
// In LanguageToolService
constructor() {
  this.debug = process.env.NODE_ENV === 'development';
  // ...
}

async checkText(text) {
  if (this.debug) {
    console.log('Spell check request:', { textLength: text.length });
  }
  // ...
}
```

This integration provides a secure, scalable solution for spell checking with JWT authentication while maintaining compatibility with your existing Lexical editor setup.