CyberSentinel Email Security Dashboard

Deployment Manual

Table of Contents

System Requirements

Minimum Requirements

- Node.js v16.0 or higher
- npm v8.0 or higher
- Next.js v14.0 or higher
- 2GB RAM (4GB recommended)
- 10GB storage space
- Modern web browser (Chrome, Firefox, Edge, Safari)

Recommended Development Environment

- Visual Studio Code with ESLint and Prettier
- Git for version control
- Node.js v18.0 or higher
- npm v9.0 or higher

Installation

Local Development Setup

- Clone the repository:

  git clone https://github.com/htmw/2025S-Tech-Titans
- cd 2025S-Tech-Titans/cybersentinel
1.
- Install dependencies:

  npm install
2.
3. Set up environment variables:

    - Create a `.env.local` file in the project root directory
    - Add the required environment variables (see Configuration section)
- Start the development server:

  npm run dev
4.
5. Access the application at http://localhost:3000

Configuration

Environment Variables

Create a `.env.local` file with the following variables:

- # ML API Key for Email Analysis
- ML_API_KEY=your_ml_api_key
-
- # Next.js Configuration
- NEXT_PUBLIC_APP_URL=http://localhost:3000
-
- # Local Storage Prefix (for multi-tenant deployment)
- NEXT_PUBLIC_STORAGE_PREFIX=cybersentinel_
-
- # Optional: Analytics Integration
- NEXT_PUBLIC_ANALYTICS_ID=your_analytics_id

Project Structure

Key directories and files:

- `/app`: Next.js app router setup
- `/components`: React components
- `/lib`: Utility functions and contexts
- `/public`: Static assets

Database Setup

CyberSentinel uses localStorage for data persistence in the default configuration. For production deployments, you should implement a proper database solution.

Setting up MongoDB (Recommended)

1. Create a MongoDB Atlas account or set up a local MongoDB server

2. Create a new database named `cybersentinel`

- Add the MongoDB connection string to your `.env.local` file:

  MONGODB_URI=your_mongodb_connection_string
3.
4. Update the storage providers in `/lib` directory to use MongoDB instead of localStorage:

    - Modify `auth-context.tsx`
    - Modify `global-storage.ts`
    - Modify `notifications-store.ts`

Security Configuration

API Key Security

1. Store your ML API key securely in environment variables
2. Never expose your API key in client-side code
3. Set up proper request validation in your API routes

Authentication

The default implementation uses a simple localStorage-based authentication system. For production:

- Implement a more secure authentication provider like NextAuth.js:

  npm install next-auth

1.
2. Configure authentication providers in `/app/api/auth/[...nextauth]/route.ts`

3. Update the `AuthProvider` component to use NextAuth

HTTPS Setup

For production deployment, always use HTTPS:

1. Obtain an SSL certificate (e.g., from Let's Encrypt)
2. Configure your hosting provider to use HTTPS
3. Update the `NEXT_PUBLIC_APP_URL` environment variable to use https://

Testing

Running Tests

Run the test suite to ensure everything is working:

- npm test

Manual Testing Checklist

- Verify user registration and login
- Test email analysis functionality with the ML API
- Check all security tools
- Verify permissions for each role
- Test notifications system
- Ensure responsive design works on various devices

Deployment

Deploying to Vercel (Recommended)

1. Create a Vercel account and connect it to your GitHub repository
2. Add all environment variables in the Vercel project settings

- Deploy using the Vercel dashboard or CLI:
  npm install -g vercelvercel
3.

Deploying to Other Platforms

For other platforms (AWS, Azure, DigitalOcean, etc.):

- Build the production version:

  npm run build
1.
- Test the production build locally:

  npm start
2.
3. Deploy the `.next` directory and necessary server files according to your hosting provider's instructions

Updating

Updating Dependencies

Regularly update dependencies to ensure security and performance:

- npm update

For major updates, follow these steps:

1. Check the changelog for breaking changes
2. Update dependencies one at a time
3. Run tests after each update
4. Fix any issues before proceeding to the next dependency

Version Control

Follow these best practices for version management:

1. Use semantic versioning (MAJOR.MINOR.PATCH)
2. Create release tags for each version
3. Maintain a changelog documenting all significant changes