🚀 AWIBI MEDTECH Comprehensive Deployment Guide

Version: 3.0
Date: July 9, 2025
Author: Manus AI
Status: Production Ready

Table of Contents

Overview

This comprehensive guide provides step-by-step instructions for deploying the AWIBI MEDTECH full-stack application to production. The application consists of a React frontend deployed on Vercel and a Node.js/Express backend deployed on Render, with MongoDB Atlas as the database.

Architecture Overview

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   Frontend      │    │    Backend      │    │    Database     │
│   (Vercel)      │◄──►│   (Render)      │◄──►│ (MongoDB Atlas) │
│   React + Vite  │    │ Node.js/Express │    │   Cloud DB      │
└─────────────────┘    └─────────────────┘    └─────────────────┘
```

Key Features

- **Frontend:** React 18 with Vite, TailwindCSS, Shadcn/UI
- **Backend:** Node.js with Express, Passport.js authentication
- **Database:** MongoDB with Mongoose ODM
- **Authentication:** JWT + Google OAuth 2.0
- **Security:** Helmet, CORS, Rate limiting, Input validation
- **Deployment:** Vercel (frontend) + Render (backend)

Prerequisites

Before beginning the deployment process, ensure you have the following accounts and tools set up. These prerequisites are essential for a successful deployment and ongoing maintenance of the AWIBI MEDTECH application.

Required Accounts

Vercel Account Create a free account at vercel.com for frontend deployment. Vercel provides excellent React application hosting with automatic deployments from Git repositories, global CDN distribution, and seamless integration with modern frontend frameworks.

Render Account Sign up for a free account at render.com for backend deployment. Render offers reliable Node.js hosting with automatic deployments, built-in SSL certificates, and excellent performance for API services.

MongoDB Atlas Account Register at mongodb.com/cloud/atlas for database hosting. MongoDB Atlas provides a fully managed cloud database service with automatic scaling, backup, and security features.

Google Cloud Console Account Access console.cloud.google.com to set up Google OAuth credentials. This is required for the Google authentication feature in the application.

GitHub Account Ensure you have a GitHub account for code repository management and automated deployments.

Required Tools

Node.js and npm Install Node.js version 18 or higher from nodejs.org. This includes npm (Node Package Manager) which is required for managing dependencies and running build scripts.

Git Install Git for version control and repository management. Download from [git-scm.com](git-scm.com).

Code Editor Use a modern code editor like Visual Studio Code, which provides excellent support for JavaScript, React, and Node.js development.

Domain Configuration (Optional)

If you plan to use a custom domain, ensure you have: - Domain name registered with a domain registrar - Access to DNS management for the domain - Understanding of DNS record types (A, CNAME, TXT)

Environment Preparation

Before deployment, verify that your local development environment is working correctly:

1. **Frontend Development Server** `bash cd frontend/awibi-medtech-frontend npm install npm run dev`

2. **Backend Development Server** `bash cd backend npm install npm start`

3. **Database Connection** Ensure you can connect to your MongoDB instance (local or cloud)

4. **Environment Variables** Verify all required environment variables are identified and documented

---

Frontend Deployment (Vercel)

Deploying the AWIBI MEDTECH frontend to Vercel is a streamlined process that leverages Vercel's excellent support for React applications and automatic deployments from Git repositories.

Step 1: Repository Preparation

First, ensure your frontend code is properly organized and committed to a Git repository. The frontend should be located in the `frontend/awibi-medtech-frontend` directory with the following structure:

```
frontend/awibi-medtech-frontend/
├── public/
├── src/
├── index.html
├── package.json
├── vite.config.js
├── tailwind.config.js
└── .env.example
```

Important Files to Verify:

1. **package.json** - Ensure all dependencies are listed and build scripts are configured: `json { "scripts": { "dev": "vite", "build": "vite build", "preview": "vite preview" } }`

2. **vite.config.js** - Verify Vite configuration is optimized for production: ```javascript import { defineConfig } from 'vite' import react from '@vitejs/plugin-react'

export default defineConfig({ plugins: [react()], build: { outDir: 'dist', sourcemap: false, minify: 'terser' } }) ```

Step 2: Environment Variables Setup

Create production environment variables that will be configured in Vercel:

Required Environment Variables: - `VITE_API_URL` - Your backend API URL (will be your Render deployment URL) - `VITE_GOOGLE_CLIENT_ID` - Google OAuth client ID for authentication - `VITE_APP_NAME` - Application name for branding - `VITE_APP_VERSION` - Application version for tracking

Step 3: Vercel Deployment Process

Method 1: Git Integration (Recommended)

1. **Connect Repository to Vercel**
2. Log in to your Vercel dashboard
3. Click "New Project"
4. Import your Git repository containing the AWIBI MEDTECH code
5. Select the repository and click "Import"
6. **Configure Project Settings**
7. **Framework Preset:** Vite
8. **Root Directory:** `frontend/awibi-medtech-frontend`
9. **Build Command:** `npm run build`
10. **Output Directory:** `dist`
11. **Install Command:** `npm install`
12. **Environment Variables Configuration** Navigate to Project Settings → Environment Variables and add: `VITE_API_URL=https://your-backend-app.onrender.com VITE_GOOGLE_CLIENT_ID=your-google-client-id VITE_APP_NAME=AWIBI MEDTECH VITE_APP_VERSION=3.0.0`
13. **Deploy** Click "Deploy" and Vercel will automatically build and deploy your application.

Method 2: Vercel CLI

1. **Install Vercel CLI** `bash npm install -g vercel`

2. **Login to Vercel** `bash vercel login`

3. **Deploy from Project Directory** `bash cd frontend/awibi-medtech-frontend vercel --prod`

Step 4: Custom Domain Configuration (Optional)

If you have a custom domain:

1. **Add Domain in Vercel**
2. Go to Project Settings → Domains
3. Add your custom domain

4. Follow DNS configuration instructions

5. **DNS Configuration** Add the following DNS records: ``` Type: CNAME Name: www Value: cname.vercel-dns.com

Type: A Name: @ Value: 76.76.19.61 ```

Step 5: Build Optimization

Ensure optimal build performance by configuring:

1. **Bundle Analysis** Add bundle analyzer to identify large dependencies: `bash npm install --save-dev rollup-plugin-visualizer`

2. **Code Splitting** Implement route-based code splitting in your React components: `javascript const LazyComponent = lazy(() => import('./Component'));`

3. **Asset Optimization** Configure Vite for optimal asset handling: `javascript export default defineConfig({ build: { rollupOptions: { output: { manualChunks: { vendor: ['react', 'react-dom'], ui: ['@radix-ui/react-dialog', '@radix-ui/react-dropdown-menu'] } } } } })`

Step 6: Deployment Verification

After deployment, verify the following:

1. **Application Loading**
2. Visit your Vercel deployment URL
3. Verify the homepage loads correctly
4. Check that all navigation links work
5. **API Connectivity**
6. Test login functionality
7. Verify API calls are reaching your backend
8. Check browser console for any CORS errors
9. **Performance Testing**
10. Use Lighthouse to audit performance
11. Verify Core Web Vitals scores
12. Test loading speed from different locations
13. **Mobile Responsiveness**
14. Test on various device sizes
15. Verify touch interactions work correctly
16. Check that all content is accessible on mobile

Common Vercel Deployment Issues and Solutions

Build Failures - Ensure all dependencies are listed in package.json - Check for TypeScript errors if using TypeScript - Verify environment variables are properly set

CORS Issues - Ensure backend CORS configuration includes your Vercel domain - Check that API URLs are correctly configured - Verify environment variables are properly set

Performance Issues - Implement code splitting for large bundles - Optimize images and assets - Use Vercel's built-in analytics to identify bottlenecks

---

Backend Deployment (Render)

Deploying the AWIBI MEDTECH backend to Render provides a robust, scalable hosting solution for the Node.js/Express API server. Render offers automatic deployments, built-in SSL, and excellent performance for backend services.

Step 1: Backend Preparation

Ensure your backend code is properly structured and optimized for production deployment. The backend should be located in the `backend` directory with the following essential files:

```
backend/
├── server-final-production.js
├── package.json
├── .env.example
├── models/
├── routes/
├── middleware/
├── config/
└── README.md
```

Critical Files to Verify:

1. **package.json** - Ensure production dependencies and scripts are configured: `json { "name": "awibi-medtech-backend", "version": "3.0.0", "main": "server-final-production.js", "scripts": { "start": "node server-final-production.js", "dev": "nodemon server-final-production.js" }, "dependencies": { "express": "^4.18.2", "mongoose": "^7.5.0", "cors": "^2.8.5", "helmet": "^7.0.0", "bcryptjs": "^2.4.3", "jsonwebtoken": "^9.0.2", "passport": "^0.6.0", "passport-google-oauth20": "^2.0.0", "express-rate-limit": "^6.10.0" } }`

2. **server-final-production.js** - Verify the main server file is configured for production: `javascript const PORT = process.env.PORT || 5000; app.listen(PORT, '0.0.0.0', () => { console.log(`🚀 Server running on port ${PORT}`); });`

Step 2: Environment Variables Configuration

Prepare all required environment variables for production deployment:

Essential Environment Variables:

```
NODE_ENV=production
PORT=5000
MONGODB_URI=mongodb+srv://username:password@cluster.mongodb.net/awibi-medtech
JWT_SECRET=your-super-secure-jwt-secret-key
JWT_EXPIRE=7d
BCRYPT_ROUNDS=12

# Google OAuth
GOOGLE_CLIENT_ID=your-google-client-id
GOOGLE_CLIENT_SECRET=your-google-client-secret

# CORS Configuration
FRONTEND_URL=https://your-vercel-app.vercel.app
ALLOWED_ORIGINS=https://your-vercel-app.vercel.app,https://your-custom-
domain.com

# Security
MAX_LOGIN_ATTEMPTS=5
ACCOUNT_LOCK_TIME=900000
PASSWORD_MIN_LENGTH=8

# Email Configuration (Optional)
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your-email@gmail.com
SMTP_PASS=your-app-password
```

Step 3: Render Deployment Process

Method 1: Git Integration (Recommended)

1. **Create New Web Service**
2. Log in to your Render dashboard
3. Click "New +" and select "Web Service"
4. Connect your GitHub repository
5. Select the repository containing your AWIBI MEDTECH code
6. **Configure Service Settings** `Name: awibi-medtech-backend Environment: Node Region: Choose closest to your users Branch: main (or your production branch) Root Directory: backend Build Command: npm install Start Command: npm start`
7. **Advanced Settings** `Auto-Deploy: Yes Health Check Path: /health`

8. **Environment Variables** Add all the environment variables listed above in the Environment section of your Render service.

Method 2: Render CLI

1. **Install Render CLI** `bash npm install -g @render/cli`

2. **Login to Render** `bash render login`

3. **Deploy from Backend Directory** `bash cd backend render deploy`

Step 4: Database Connection Setup

Configure MongoDB Atlas connection for production:

1. **MongoDB Atlas Cluster**
2. Create a new cluster in MongoDB Atlas
3. Configure network access to allow connections from anywhere (0.0.0.0/0)

4. Create a database user with read/write permissions

5. **Connection String** `mongodb+srv://<username>:<password>@<cluster-name>.mongodb.net/<database-name>?retryWrites=true&w=majority`

6. **Database Configuration** Ensure your database configuration file handles production connections: `` javascript const connectDB = async () => { try { const conn = await mongoose.connect(process.env.MONGODB_URI, { useNewUrlParser: true, useUnifiedTopology: true, }); console.log(`MongoDB Connected: ${conn.connection.host}`); } catch (error) { console.error('Database connection error:', error); process.exit(1); } }; ``

Step 5: CORS Configuration for Production

Configure CORS to work with your Vercel frontend:

```
  const corsOptions = {
    origin: function (origin, callback) {
      const allowedOrigins = process.env.ALLOWED_ORIGINS?.split(',') || [
        'https://your-vercel-app.vercel.app',
        'https://your-custom-domain.com'
      ];

      // Allow requests with no origin (mobile apps, etc.)
      if (!origin) return callback(null, true);

      if (allowedOrigins.indexOf(origin) !== -1) {
        callback(null, true);
      } else {
        callback(new Error('Not allowed by CORS'));
      }
    },
    credentials: true,
    methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS', 'PATCH'],
    allowedHeaders: [
      'Content-Type',
      'Authorization',
      'X-Requested-With',
      'Accept',
      'Origin'
    ],
    optionsSuccessStatus: 200
};

app.use(cors(corsOptions));
```

Step 6: Security Configuration

Implement production-ready security measures:

1. **Helmet Configuration** `javascript app.use(helmet({ contentSecurityPolicy: { directives: { defaultSrc: ["'self'"], styleSrc: ["'self'", "'unsafe-inline'"], scriptSrc: ["'self'"], imgSrc: ["'self'", "data:", "https:"], }, }, hsts: { maxAge: 31536000, includeSubDomains: true, preload: true } }));`

2. **Rate Limiting** `javascript const limiter = rateLimit({ windowMs: 15 * 60 * 1000, // 15 minutes max: 100, // limit each IP to 100 requests per windowMs message: 'Too many requests from this IP' }); app.use('/api/', limiter);`

3. **Input Validation** `javascript app.use(express.json({ limit: '10mb' })); app.use(express.urlencoded({ extended: true, limit: '10mb' })); app.use(mongoSanitize()); app.use(xss());`

Step 7: Health Check Endpoint

Implement a comprehensive health check for Render monitoring:

```
app.get('/health', async (req, res) => {
  try {
    // Check database connection
    const dbStatus = mongoose.connection.readyState === 1 ? 'Connected' :
'Disconnected';

    // Check memory usage
    const memoryUsage = process.memoryUsage();

    res.status(200).json({
      status: 'OK',
      timestamp: new Date().toISOString(),
      uptime: process.uptime(),
      environment: process.env.NODE_ENV,
      database: dbStatus,
      memory: {
        rss: Math.round(memoryUsage.rss / 1024 / 1024) + ' MB',
        heapUsed: Math.round(memoryUsage.heapUsed / 1024 / 1024) + ' MB'
      },
      version: '3.0.0'
    });
  } catch (error) {
    res.status(500).json({
      status: 'ERROR',
      message: error.message
    });
  }
});
```

Step 8: Deployment Verification

After deployment, thoroughly test your backend:

1. **Health Check** `bash curl https://your-render-app.onrender.com/health`

2. **API Endpoints** Test critical endpoints: ```bash # Test CORS curl -H "Origin: https://your-vercel-app.vercel.app" \ https://your-render-app.onrender.com/api/test-cors

# Test Authentication curl -X POST \ -H "Content-Type: application/json" \ -d '{"email":"test@example.com","password":"test123"}' \ https://your-render-app.onrender.com/api/auth/test-login ```

1. **Database Connectivity** Verify database operations work correctly through your API endpoints.

2. **Performance Testing** Use tools like Apache Bench or Artillery to test API performance under load.

Common Render Deployment Issues and Solutions

Build Failures - Ensure package.json includes all production dependencies - Check Node.js version compatibility - Verify build commands are correct

Database Connection Issues - Verify MongoDB Atlas network access settings - Check connection string format and credentials - Ensure database user has proper permissions

Environment Variable Issues - Double-check all required environment variables are set - Verify sensitive values are properly escaped - Test environment variable access in your application

Memory and Performance Issues - Monitor memory usage through Render dashboard - Implement proper error handling to prevent memory leaks - Consider upgrading to a higher-tier Render plan for better performance

---

Environment Configuration

Proper environment configuration is crucial for the successful deployment and operation of the AWIBI MEDTECH application. This section provides detailed guidance on setting up environment variables for both development and production environments.

Frontend Environment Variables

The frontend application uses Vite, which requires environment variables to be prefixed with `VITE_` to be accessible in the browser. Create the following environment files:

Development Environment (.env)

```
# API Configuration
VITE_API_URL=http://localhost:5000

# Google OAuth Configuration
VITE_GOOGLE_CLIENT_ID=your-development-google-client-id

# Application Configuration
VITE_APP_NAME=AWIBI MEDTECH
VITE_APP_VERSION=3.0.0
VITE_APP_ENVIRONMENT=development

# Feature Flags
VITE_ENABLE_ANALYTICS=false
VITE_ENABLE_DEBUG=true
VITE_ENABLE_MOCK_DATA=true
```

Production Environment (.env.production)

```
# API Configuration
VITE_API_URL=https://your-backend-app.onrender.com

# Google OAuth Configuration
VITE_GOOGLE_CLIENT_ID=your-production-google-client-id

# Application Configuration
VITE_APP_NAME=AWIBI MEDTECH
VITE_APP_VERSION=3.0.0
VITE_APP_ENVIRONMENT=production

# Feature Flags
VITE_ENABLE_ANALYTICS=true
VITE_ENABLE_DEBUG=false
VITE_ENABLE_MOCK_DATA=false

# Performance Configuration
VITE_ENABLE_PWA=true
VITE_CACHE_DURATION=3600
```

Environment Variable Usage in Code

```
// API configuration
const API_URL = import.meta.env.VITE_API_URL;

// Feature flags
const isDebugEnabled = import.meta.env.VITE_ENABLE_DEBUG === 'true';

// Application info
const appVersion = import.meta.env.VITE_APP_VERSION;
```

Backend Environment Variables

The backend requires comprehensive environment configuration for database connections, authentication, security, and external service integrations.

Development Environment (.env)

```
# Application Configuration
NODE_ENV=development
PORT=5000
APP_NAME=AWIBI MEDTECH Backend
APP_VERSION=3.0.0

# Database Configuration
MONGODB_URI=mongodb://localhost:27017/awibi-medtech-dev
DB_NAME=awibi-medtech-dev

# JWT Configuration
JWT_SECRET=your-development-jwt-secret-key-minimum-32-characters
JWT_EXPIRE=7d
JWT_REFRESH_EXPIRE=30d

# Password Security
BCRYPT_ROUNDS=10
PASSWORD_MIN_LENGTH=8
MAX_LOGIN_ATTEMPTS=5
ACCOUNT_LOCK_TIME=900000

# Google OAuth Configuration
GOOGLE_CLIENT_ID=your-development-google-client-id
GOOGLE_CLIENT_SECRET=your-development-google-client-secret

# CORS Configuration
FRONTEND_URL=http://localhost:5173
ALLOWED_ORIGINS=http://localhost:5173,http://localhost:3000

# Email Configuration (Development)
SMTP_HOST=smtp.mailtrap.io
SMTP_PORT=2525
SMTP_USER=your-mailtrap-username
SMTP_PASS=your-mailtrap-password
FROM_EMAIL=noreply@awibi-medtech.com
FROM_NAME=AWIBI MEDTECH

# File Upload Configuration
MAX_FILE_SIZE=5242880
ALLOWED_FILE_TYPES=image/jpeg,image/png,image/gif,application/pdf

# Rate Limiting
RATE_LIMIT_WINDOW=900000
RATE_LIMIT_MAX_REQUESTS=100
AUTH_RATE_LIMIT_MAX=5

# Session Configuration
SESSION_SECRET=your-session-secret-key
SESSION_EXPIRE=86400000

# Logging Configuration
LOG_LEVEL=debug
LOG_FILE=logs/app.log
```

Production Environment (.env.production)

```
# Application Configuration
NODE_ENV=production
PORT=5000
APP_NAME=AWIBI MEDTECH Backend
APP_VERSION=3.0.0

# Database Configuration
MONGODB_URI=mongodb+srv://username:password@cluster.mongodb.net/awibi-medtech?
retryWrites=true&w=majority
DB_NAME=awibi-medtech

# JWT Configuration
JWT_SECRET=your-super-secure-production-jwt-secret-key-minimum-64-characters
JWT_EXPIRE=7d
JWT_REFRESH_EXPIRE=30d

# Password Security
BCRYPT_ROUNDS=12
PASSWORD_MIN_LENGTH=8
MAX_LOGIN_ATTEMPTS=5
ACCOUNT_LOCK_TIME=900000

# Google OAuth Configuration
GOOGLE_CLIENT_ID=your-production-google-client-id
GOOGLE_CLIENT_SECRET=your-production-google-client-secret

# CORS Configuration
FRONTEND_URL=https://your-vercel-app.vercel.app
ALLOWED_ORIGINS=https://your-vercel-app.vercel.app,https://your-custom-
domain.com

# Email Configuration (Production)
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your-gmail-address@gmail.com
SMTP_PASS=your-gmail-app-password
FROM_EMAIL=noreply@awibi-medtech.com
FROM_NAME=AWIBI MEDTECH

# File Upload Configuration
MAX_FILE_SIZE=5242880
ALLOWED_FILE_TYPES=image/jpeg,image/png,image/gif,application/pdf

# Rate Limiting
RATE_LIMIT_WINDOW=900000
RATE_LIMIT_MAX_REQUESTS=100
AUTH_RATE_LIMIT_MAX=5

# Session Configuration
SESSION_SECRET=your-production-session-secret-key
SESSION_EXPIRE=86400000

# Logging Configuration
LOG_LEVEL=info
LOG_FILE=logs/app.log

# Security Configuration
HELMET_CSP_ENABLED=true
TRUST_PROXY=true

# Monitoring Configuration
```

```
HEALTH_CHECK_ENABLED=true
METRICS_ENABLED=true
```

Environment Variable Security Best Practices

Secret Generation Generate secure secrets using cryptographically strong methods:

```
# Generate JWT secret (64 characters)
node -e "console.log(require('crypto').randomBytes(32).toString('hex'))"

# Generate session secret (32 characters)
node -e "console.log(require('crypto').randomBytes(16).toString('hex'))"

# Generate API key (UUID format)
node -e "console.log(require('crypto').randomUUID())"
```

Environment Variable Validation Implement validation in your application to ensure all required environment variables are present:

```
const requiredEnvVars = [
  'NODE_ENV',
  'PORT',
  'MONGODB_URI',
  'JWT_SECRET',
  'GOOGLE_CLIENT_ID',
  'GOOGLE_CLIENT_SECRET',
  'FRONTEND_URL'
];

const validateEnvironment = () => {
  const missingVars = requiredEnvVars.filter(varName => !process.env[varName]);

  if (missingVars.length > 0) {
    console.error('Missing required environment variables:', missingVars);
    process.exit(1);
  }

  // Validate JWT secret length
  if (process.env.JWT_SECRET.length < 32) {
    console.error('JWT_SECRET must be at least 32 characters long');
    process.exit(1);
  }

  console.log('✅ Environment validation passed');
};

validateEnvironment();
```

Platform-Specific Configuration

Vercel Environment Variables Configure in Vercel dashboard under Project Settings → Environment Variables:

1. **Production Variables**
2. Set for "Production" environment
3. Include all VITE_ prefixed variables
4. Ensure API_URL points to your Render backend
5. **Preview Variables**
6. Set for "Preview" environment
7. Use staging/development values
8. Useful for testing before production deployment
9. **Development Variables**
10. Set for "Development" environment
11. Use local development values

Render Environment Variables Configure in Render dashboard under your service → Environment:

1. **Sensitive Variables**
2. Mark database passwords and API keys as sensitive
3. These will be hidden in the dashboard after creation
4. **Auto-Deploy Configuration**
5. Changes to environment variables trigger automatic redeployment
6. Plan deployments during low-traffic periods
7. **Environment Groups**
8. Create environment groups for shared variables
9. Useful for managing multiple services with common configuration

Environment Configuration Testing

Development Testing

```
# Test environment variable loading
cd backend
node -e "
require('dotenv').config();
console.log('NODE_ENV:', process.env.NODE_ENV);
console.log('MONGODB_URI:', process.env.MONGODB_URI ? 'Set' : 'Missing');
console.log('JWT_SECRET:', process.env.JWT_SECRET ? 'Set' : 'Missing');
"
```

Production Testing

```
# Test production environment variables
curl https://your-backend-app.onrender.com/health

# Verify CORS configuration
curl -H "Origin: https://your-frontend-app.vercel.app" \
     https://your-backend-app.onrender.com/api/test-cors
```

Environment Migration Checklist

When moving from development to production:

- [ ] Update all URLs from localhost to production domains
- [ ] Generate new, secure secrets for production
- [ ] Configure production database connection
- [ ] Set up production email service
- [ ] Update Google OAuth redirect URIs
- [ ] Configure production CORS origins
- [ ] Enable production security features
- [ ] Set appropriate log levels
- [ ] Configure monitoring and health checks
- [ ] Test all environment variables are accessible
- [ ] Verify application functionality with production configuration

---

CORS Configuration

Cross-Origin Resource Sharing (CORS) configuration is critical for the AWIBI MEDTECH application to function properly in production. This section provides comprehensive guidance on implementing robust CORS policies that ensure security while enabling seamless communication between the frontend and backend.

Understanding CORS in the AWIBI MEDTECH Context

The AWIBI MEDTECH application operates with a distributed architecture where the React frontend (hosted on Vercel) needs to communicate with the Node.js backend (hosted on Render). This cross-origin communication requires careful CORS configuration to prevent security vulnerabilities while maintaining functionality.

Architecture Overview:

```
 Frontend (Vercel)          Backend (Render)
https://app.vercel.app  →  https://api.onrender.com
Origin: vercel.app         CORS Policy: Allow vercel.app
```

Production CORS Configuration

The backend implements a sophisticated CORS configuration that balances security with functionality:

```javascript
const corsOptions = {
  origin: function (origin, callback) {
    // Define allowed origins from environment variables
    const allowedOrigins = process.env.ALLOWED_ORIGINS?.split(',') || [
      'https://awibi-medtech.vercel.app',
      'https://www.awibi-medtech.com',
      'https://awibi-medtech.com'
    ];

    // Allow requests with no origin (mobile apps, Postman, etc.)
    if (!origin) {
      return callback(null, true);
    }

    // Check if the origin is in the allowed list
    if (allowedOrigins.includes(origin)) {
      callback(null, true);
    } else {
      console.warn(`CORS blocked request from origin: ${origin}`);
      callback(new Error(`Origin ${origin} not allowed by CORS policy`));
    }
  },

  // Enable credentials for authentication
  credentials: true,

  // Allowed HTTP methods
  methods: [
    'GET',
    'POST',
    'PUT',
    'DELETE',
    'OPTIONS',
    'PATCH',
    'HEAD'
  ],

  // Allowed headers
  allowedHeaders: [
    'Content-Type',
    'Authorization',
    'X-Requested-With',
    'Accept',
    'Origin',
    'Cache-Control',
    'X-File-Name',
    'X-CSRF-Token'
  ],

  // Exposed headers (accessible to frontend)
  exposedHeaders: [
    'X-Total-Count',
    'X-Page-Count',
    'X-Rate-Limit-Remaining',
    'X-Rate-Limit-Reset'
  ],

  // Preflight cache duration (24 hours)
  maxAge: 86400,

  // Success status for legacy browsers
  optionsSuccessStatus: 200,

  // Disable preflight for simple requests
  preflightContinue: false
};

// Apply CORS middleware
app.use(cors(corsOptions));
```

Development CORS Configuration

For development environments, a more permissive CORS configuration is used to facilitate testing and development:

```
const developmentCorsOptions = {
  origin: function (origin, callback) {
    // In development, allow common development origins
    const developmentOrigins = [
      'http://localhost:3000',
      'http://localhost:5173',
      'http://localhost:5174',
      'http://127.0.0.1:3000',
      'http://127.0.0.1:5173',
      'http://127.0.0.1:5174'
    ];

    // Allow requests with no origin
    if (!origin) return callback(null, true);

    // Allow all development origins
    if (developmentOrigins.includes(origin) || origin.includes('localhost')) {
      callback(null, true);
    } else {
      callback(null, true); // More permissive in development
    }
  },
  credentials: true,
  methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS', 'PATCH'],
  allowedHeaders: ['Content-Type', 'Authorization', 'X-Requested-With',
'Accept', 'Origin'],
  optionsSuccessStatus: 200
};

// Use development CORS in development environment
if (process.env.NODE_ENV === 'development') {
  app.use(cors(developmentCorsOptions));
} else {
  app.use(cors(corsOptions));
}
```

Dynamic CORS Configuration

For applications that need to support multiple environments or dynamic origin management:

```
class CORSManager {
  constructor() {
    this.allowedOrigins = new Set();
    this.loadAllowedOrigins();
  }

  loadAllowedOrigins() {
    // Load from environment variables
    const envOrigins = process.env.ALLOWED_ORIGINS?.split(',') || [];
    envOrigins.forEach(origin => this.allowedOrigins.add(origin.trim()));

    // Add default origins based on environment
    if (process.env.NODE_ENV === 'production') {
      this.allowedOrigins.add('https://awibi-medtech.vercel.app');
      this.allowedOrigins.add('https://www.awibi-medtech.com');
    } else {
      this.allowedOrigins.add('http://localhost:5173');
      this.allowedOrigins.add('http://localhost:3000');
    }
  }

  isOriginAllowed(origin) {
    if (!origin) return true; // Allow requests with no origin

    // Check exact match
    if (this.allowedOrigins.has(origin)) return true;

    // Check subdomain patterns for production
    if (process.env.NODE_ENV === 'production') {
      const allowedDomains = ['vercel.app', 'awibi-medtech.com'];
      return allowedDomains.some(domain =>
        origin.endsWith(`.$`{domain}`) || origin === `https://`${domain}`
      );
    }

    // Allow localhost in development
    if (process.env.NODE_ENV === 'development') {
      return origin.includes('localhost') || origin.includes('127.0.0.1');
    }

    return false;
  }

  getCorsOptions() {
    return {
      origin: (origin, callback) => {
        if (this.isOriginAllowed(origin)) {
          callback(null, true);
        } else {
          console.warn(`CORS: Blocked request from ${origin}`);
          callback(new Error('Not allowed by CORS'));
        }
      },
      credentials: true,
      methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS', 'PATCH'],
      allowedHeaders: [
        'Content-Type',
        'Authorization',
        'X-Requested-With',
        'Accept',
        'Origin'
      ],
      optionsSuccessStatus: 200
    };
  }
}

const corsManager = new CORSManager();
app.use(cors(corsManager.getCorsOptions()));
```

Frontend CORS Handling

The frontend needs to be configured to work properly with the CORS-enabled backend:

```
 // API client configuration
const apiClient = axios.create({
  baseURL: import.meta.env.VITE_API_URL,
  withCredentials: true, // Important for CORS with credentials
  timeout: 10000,
  headers: {
    'Content-Type': 'application/json',
    'Accept': 'application/json'
  }
});

// Request interceptor for authentication
apiClient.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('authToken');
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);

// Response interceptor for error handling
apiClient.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      // Handle authentication errors
      localStorage.removeItem('authToken');
      window.location.href = '/login';
    }

    if (error.code === 'ERR_NETWORK') {
      console.error('Network error - possible CORS issue');
    }

    return Promise.reject(error);
  }
);
```

CORS Testing and Validation

Manual CORS Testing

```
 # Test basic CORS functionality
curl -H "Origin: https://your-frontend-domain.vercel.app" \
     -H "Access-Control-Request-Method: POST" \
     -H "Access-Control-Request-Headers: Content-Type,Authorization" \
     -X OPTIONS \
     https://your-backend-domain.onrender.com/api/auth/login

# Test actual API request with CORS
curl -H "Origin: https://your-frontend-domain.vercel.app" \
     -H "Content-Type: application/json" \
     -X POST \
     -d '{"email":"test@example.com","password":"test123"}' \
     https://your-backend-domain.onrender.com/api/auth/test-login
```

Automated CORS Testing

```
 // CORS test suite
const testCORS = async () => {
  const testOrigins = [
    'https://awibi-medtech.vercel.app',
    'https://malicious-site.com',
    'http://localhost:5173'
  ];

  for (const origin of testOrigins) {
    try {
      const response = await fetch(`${API_URL}/api/test-cors`, {
        method: 'GET',
        headers: {
          'Origin': origin
        }
      });

      console.log(`Origin $`{origin}: `${response.ok ? 'ALLOWED' :
'BLOCKED'}`);
    } catch (error) {
      console.log(`Origin $`{origin}: BLOCKED (`${error.message})`);
    }
  }
};
```

Common CORS Issues and Solutions

Issue 1: Preflight Request Failures

```
 Error: Access to fetch at 'api.onrender.com' from origin 'app.vercel.app'
has been blocked by CORS policy: Response to preflight request doesn't
pass access control check
```

Solution:

```
 // Ensure OPTIONS method is handled
app.options('*', cors(corsOptions)); // Enable preflight for all routes

// Or handle preflight manually
app.use((req, res, next) => {
  if (req.method === 'OPTIONS') {
    res.header('Access-Control-Allow-Origin', req.headers.origin);
    res.header('Access-Control-Allow-Methods', 'GET,POST,PUT,DELETE,OPTIONS');
    res.header('Access-Control-Allow-Headers', 'Content-Type,Authorization');
    res.header('Access-Control-Allow-Credentials', 'true');
    return res.sendStatus(200);
  }
  next();
});
```

Issue 2: Credentials Not Included

```
Error: Cannot read property 'user' of undefined
```

Solution:

```
 // Backend: Ensure credentials are allowed
app.use(cors({
  credentials: true,
  origin: allowedOrigins
}));

// Frontend: Include credentials in requests
fetch('/api/user', {
  credentials: 'include'
});

// Or with axios
axios.defaults.withCredentials = true;
```

Issue 3: Dynamic Subdomain Support

```
Error: Subdomain not allowed by CORS policy
```

Solution:

```
const corsOptions = {
  origin: (origin, callback) => {
    if (!origin) return callback(null, true);

    // Allow main domain and all subdomains
    const allowedDomainPattern = /^https:\/\/([a-z0-9-]+\.)?awibi-
medtech\.com$/;
    const allowedVercelPattern = /^https:\/\/[a-z0-9-]+\.vercel\.app$/;

    if (allowedDomainPattern.test(origin) || allowedVercelPattern.test(origin))
{
      callback(null, true);
    } else {
      callback(new Error('Not allowed by CORS'));
    }
  },
  credentials: true
};
```

CORS Security Best Practices

1. Principle of Least Privilege Only allow the minimum necessary origins, methods, and headers:

```
const corsOptions = {
  origin: specificAllowedOrigins, // Never use '*' in production
  methods: ['GET', 'POST'], // Only allow necessary methods
  allowedHeaders: ['Content-Type', 'Authorization'], // Minimal headers
  credentials: true
};
```

2. Environment-Specific Configuration Use different CORS policies for different environments:

```
const getCorsOptions = () => {
  if (process.env.NODE_ENV === 'production') {
    return productionCorsOptions;
  } else if (process.env.NODE_ENV === 'staging') {
    return stagingCorsOptions;
  } else {
    return developmentCorsOptions;
  }
};
```

3. Regular Security Audits Implement logging and monitoring for CORS violations:

```
const corsOptions = {
  origin: (origin, callback) => {
    if (isAllowedOrigin(origin)) {
      callback(null, true);
    } else {
      // Log security violation
      console.warn(`CORS violation: $`{origin} attempted access at `${new
Date()}`);

      // Optional: Send alert to monitoring system
      if (process.env.NODE_ENV === 'production') {
        sendSecurityAlert(`CORS violation from ${origin}`);
      }

      callback(new Error('Not allowed by CORS'));
    }
  }
};
```

CORS Monitoring and Debugging

Production Monitoring

```
// CORS monitoring middleware
app.use((req, res, next) => {
  const origin = req.headers.origin;

  // Log all cross-origin requests
  if (origin && origin !== req.headers.host) {
    console.log(`CORS request: $`{req.method} `${req.path} from ${origin}`);
  }

  // Monitor preflight requests
  if (req.method === 'OPTIONS') {
    console.log(`Preflight request from $`{origin} for `${req.headers['access-
control-request-method']}`);
  }

  next();
});
```

Debug Mode

```
if (process.env.CORS_DEBUG === 'true') {
  app.use((req, res, next) => {
    console.log('CORS Debug Info:', {
      origin: req.headers.origin,
      method: req.method,
      path: req.path,
      headers: req.headers
    });
    next();
  });
}
```

This comprehensive CORS configuration ensures that the AWIBI MEDTECH application maintains security while providing seamless cross-origin communication between the frontend and backend components.
