# RIMSS E-Commerce Platform - Non-Functional Requirements Implementation

## 1. Performance Requirements

### 1.1 Response Time

* **Requirement**: Page load time < 2 seconds
* **Implementation**:
* // LoadingState Component  
  const LoadingState: React.FC = () => {  
   const [showLoader, setShowLoader] = useState(false);  
    
   useEffect(() => {  
   const timer = setTimeout(() => {  
   setShowLoader(true);  
   }, 100);  
   return () => clearTimeout(timer);  
   }, []);  
    
   return showLoader ? <Spinner /> : null;  
  };

### 1.2 Caching Strategy

* **Requirement**: Optimize repeat visits
* **Implementation**:
* // Service Worker Registration  
  if ('serviceWorker' in navigator) {  
   window.addEventListener('load', () => {  
   navigator.serviceWorker.register('/service-worker.js');  
   });  
  }  
    
  // Cache Configuration  
  const CACHE\_NAME = 'rimss-cache-v1';  
  const urlsToCache = [  
   '/',  
   '/static/css/main.css',  
   '/static/js/main.js'  
  ];

### 1.3 Database Performance

* **Requirement**: Query response < 100ms
* **Implementation**:
* // MongoDB Indexing  
  const ProductSchema = new Schema({  
   name: { type: String, index: true },  
   category: { type: String, index: true },  
   price: { type: Number, index: true }  
  });  
    
  // Query Optimization  
  const getProducts = async (filters: ProductFilters) => {  
   return Product.find(filters)  
   .lean()  
   .select('name price category')  
   .limit(20);  
  };

## 2. Scalability Requirements

### 2.1 Horizontal Scaling

* **Requirement**: Support 1000+ concurrent users
* **Implementation**:
* // Load Balancer Configuration  
  const cluster = require('cluster');  
  const numCPUs = require('os').cpus().length;  
    
  if (cluster.isMaster) {  
   for (let i = 0; i < numCPUs; i++) {  
   cluster.fork();  
   }  
  } else {  
   // Express app  
   const app = express();  
  }

### 2.2 Database Scaling

* **Requirement**: Handle 10000+ products
* **Implementation**:
* // MongoDB Connection Pool  
  mongoose.connect(MONGODB\_URI, {  
   poolSize: 10,  
   serverSelectionTimeoutMS: 5000,  
   socketTimeoutMS: 45000  
  });

## 3. Security Requirements

### 3.1 Authentication

* **Requirement**: Secure user authentication
* **Implementation**:
* // JWT Authentication  
  const generateToken = (user: IUser): string => {  
   return jwt.sign(  
   { id: user.\_id, role: user.role },  
   process.env.JWT\_SECRET,  
   { expiresIn: '24h' }  
   );  
  };  
    
  // Password Hashing  
  const hashPassword = async (password: string): Promise<string> => {  
   const salt = await bcrypt.genSalt(10);  
   return bcrypt.hash(password, salt);  
  };

### 3.2 Data Protection

* **Requirement**: Encrypt sensitive data
* **Implementation**:
* // Data Encryption  
  const encrypt = (data: string): string => {  
   const cipher = crypto.createCipher('aes-256-cbc', process.env.ENCRYPTION\_KEY);  
   return cipher.update(data, 'utf8', 'hex') + cipher.final('hex');  
  };  
    
  // HTTPS Configuration  
  const httpsOptions = {  
   key: fs.readFileSync('key.pem'),  
   cert: fs.readFileSync('cert.pem')  
  };

## 4. Reliability Requirements

### 4.1 Error Handling

* **Requirement**: Graceful error recovery
* **Implementation**:
* // Global Error Handler  
  const errorHandler = (  
   err: AppError,  
   req: Request,  
   res: Response,  
   next: NextFunction  
  ) => {  
   logger.error(err.stack);  
   res.status(err.statusCode || 500).json({  
   status: 'error',  
   message: err.message  
   });  
  };  
    
  // API Error Boundaries  
  class ErrorBoundary extends React.Component {  
   componentDidCatch(error: Error, info: React.ErrorInfo) {  
   logger.error(error);  
   this.setState({ hasError: true });  
   }  
  }

### 4.2 Data Backup

* **Requirement**: Daily backups
* **Implementation**:
* // Automated Backup  
  const backup = async () => {  
   const timestamp = new Date().toISOString();  
   const backupPath = `backup\_${timestamp}.gz`;  
    
   await mongodump({  
   uri: MONGODB\_URI,  
   out: backupPath  
   });  
  };  
    
  // Schedule Backup  
  cron.schedule('0 0 \* \* \*', backup);

## 5. Maintainability Requirements

### 5.1 Code Organization

* **Requirement**: Modular architecture
* **Implementation**:
* // Feature-based Structure  
  src/  
   features/  
   auth/  
   components/  
   services/  
   types/  
   index.ts  
   products/  
   components/  
   services/  
   types/  
   index.ts

### 5.2 Logging

* **Requirement**: Comprehensive logging
* **Implementation**:
* // Winston Logger Configuration  
  const logger = winston.createLogger({  
   level: 'info',  
   format: winston.format.json(),  
   transports: [  
   new winston.transports.File({ filename: 'error.log', level: 'error' }),  
   new winston.transports.File({ filename: 'combined.log' })  
   ]  
  });  
    
  // Request Logging  
  app.use(morgan('combined', {  
   stream: { write: message => logger.info(message.trim()) }  
  }));

## 6. Usability Requirements

### 6.1 Responsive Design

* **Requirement**: Mobile-first approach
* **Implementation**:
* // Tailwind Configuration  
  module.exports = {  
   theme: {  
   screens: {  
   sm: '640px',  
   md: '768px',  
   lg: '1024px',  
   xl: '1280px'  
   }  
   }  
  };  
    
  // Responsive Component  
  const ProductGrid: React.FC = () => {  
   return (  
   <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3 gap-4">  
   {products.map(product => (  
   <ProductCard key={product.id} product={product} />  
   ))}  
   </div>  
   );  
  };

### 6.2 Accessibility

* **Requirement**: WCAG 2.1 compliance
* **Implementation**:
* // Accessible Components  
  const Button: React.FC<ButtonProps> = ({ children, onClick, disabled }) => {  
   return (  
   <button  
   onClick={onClick}  
   disabled={disabled}  
   aria-disabled={disabled}  
   className="focus:ring-2 focus:ring-offset-2"  
   >  
   {children}  
   </button>  
   );  
  };

## 7. SEO Requirements

### 7.1 Meta Tags

* **Requirement**: SEO optimization
* **Implementation**:
* // React Helmet Implementation  
  const SEO: React.FC<SEOProps> = ({ title, description }) => {  
   return (  
   <Helmet>  
   <title>{title}</title>  
   <meta name="description" content={description} />  
   <meta property="og:title" content={title} />  
   <meta property="og:description" content={description} />  
   </Helmet>  
   );  
  };

### 7.2 Sitemap

* **Requirement**: Dynamic sitemap
* **Implementation**:
* // Sitemap Generator  
  const generateSitemap = async () => {  
   const products = await Product.find().select('slug updatedAt');  
    
   const sitemap = new SitemapStream({  
   hostname: 'https://rimss.com'  
   });  
    
   products.forEach(product => {  
   sitemap.write({  
   url: `/product/${product.slug}`,  
   lastmod: product.updatedAt  
   });  
   });  
  };

## 8. Monitoring Requirements

### 8.1 Performance Monitoring

* **Requirement**: Real-time metrics
* **Implementation**:
* // Performance Monitoring  
  const monitorPerformance = () => {  
   const metrics = {  
   memory: process.memoryUsage(),  
   cpu: process.cpuUsage(),  
   uptime: process.uptime()  
   };  
    
   prometheus.gauge('system\_metrics').set(metrics);  
  };  
    
  setInterval(monitorPerformance, 5000);

### 8.2 Error Tracking

* **Requirement**: Error reporting
* **Implementation**:
* // Error Tracking  
  const trackError = (error: Error) => {  
   Sentry.captureException(error);  
    
   logger.error({  
   message: error.message,  
   stack: error.stack,  
   timestamp: new Date().toISOString()  
   });  
  };

## 9. Testing Requirements

### 9.1 Unit Testing

* **Requirement**: 80% code coverage
* **Implementation**:
* // Jest Configuration  
  module.exports = {  
   preset: 'ts-jest',  
   testEnvironment: 'node',  
   coverageThreshold: {  
   global: {  
   branches: 80,  
   functions: 80,  
   lines: 80,  
   statements: 80  
   }  
   }  
  };  
    
  // Example Test  
  describe('Product Service', () => {  
   it('should create product', async () => {  
   const product = await ProductService.create(mockProduct);  
   expect(product).toHaveProperty('\_id');  
   });  
  });

### 9.2 Integration Testing

* **Requirement**: API testing
* **Implementation**:
* // Supertest Implementation  
  describe('Product API', () => {  
   it('should get products', async () => {  
   const response = await request(app)  
   .get('/api/products')  
   .expect(200);  
    
   expect(response.body).toHaveProperty('products');  
   });  
  });

## 10. Documentation Requirements

### 10.1 API Documentation

* **Requirement**: OpenAPI documentation
* **Implementation**:
* // Swagger Configuration  
  const swaggerOptions = {  
   definition: {  
   openapi: '3.0.0',  
   info: {  
   title: 'RIMSS API',  
   version: '1.0.0'  
   }  
   },  
   apis: ['./src/routes/\*.ts']  
  };  
    
  app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerOptions));

### 10.2 Code Documentation

* **Requirement**: JSDoc documentation
* **Implementation**:
* /\*\*  
   \* Creates a new product in the database  
   \* @param {ProductCreationDTO} data - The product data  
   \* @returns {Promise<Product>} The created product  
   \* @throws {ValidationError} If the data is invalid  
   \*/  
  const createProduct = async (data: ProductCreationDTO): Promise<Product> => {  
   const product = new Product(data);  
   return product.save();  
  };