**Confirmation of Understanding and Acceptance of Security Requirements**

**Bidder:** ……………………………………………………………………………………………………………..……………….…………………  
**Authorized Representative (Name and Position):** ………………………………………………………………………….…………………  
**Date of Completion [dd/mm/yyyy]:** ………………………………………………………………………………..……………….……………….

By signing below, we confirm that we have **read, understood, and accepted the security requirements** detailed herein and that we **commit to complying** with these requirements throughout the execution of the project, should the contract be awarded.

**1. Access Control and Authorization:**

* **CMS Role-Based Access Control (RBAC):**
  + Implement granular RBAC within the CMS to define and enforce user permissions.
  + Roles must be configurable to align with business functions (e.g., content editor, product manager, administrator, marketing).
  + Each role must have a clearly defined set of privileges, adhering to the principle of least privilege. For example:
    - Content Editor: Create, edit, and publish content within specific sections.
    - Product Manager: Manage product catalogs, pricing, and inventory.
    - Administrator: Full access to CMS configuration, user management, and system settings.
    - Marketing: Create and manage marketing campaigns, promotions, and analytics.
  + Regularly review and update roles and permissions to reflect changes in job functions or security policies. This includes automated reviews using scripting and tools.
  + Use a framework like Spring Security (for Java-based CMS) or similar for other languages to implement RBAC.
* **Principle of Least Privilege:**
  + Enforce the principle of least privilege across the entire solution. Users and processes should only have the minimum necessary access to perform their required functions.
  + This includes database access, file system permissions, and API access. For example:
    - Database access: Use separate database accounts for the web application and the CMS, with only the required permissions (e.g., SELECT, INSERT, UPDATE on specific tables). Avoid using the root or dbo account.
    - File system permissions: The web server should run with the lowest possible privileges (e.g., a dedicated user account with restricted file system access). Limit write permissions to only necessary directories (e.g., for uploads).
    - API access: Use API keys or OAuth 2.0 with scopes to limit the actions that a client application can perform.
* **Content Review and Approval Workflow:**
  + Establish a mandatory content review and approval workflow within the CMS.
  + Different roles should be defined for content creation, review, and approval. For example:
    - Creator -> Reviewer -> Approver -> Publisher
  + The workflow should enforce separation of duties, preventing a single user from publishing content without review.
  + All content changes must be logged and auditable, including the user, timestamp, and the specific changes made. CMS frameworks like Drupal with specific plugins can implement this.
* **Authentication:**
  + Implement strong authentication mechanisms:
    - Multi-Factor Authentication (MFA) for all administrative access to the CMS, e-shop backend, and Azure portal. Consider using Azure AD MFA with Conditional Access policies. Enforce MFA for:
      * All users in the Administrator role.
      * Users accessing sensitive data or functions.
      * Remote access to the systems.
    - Strong password policies (minimum complexity, length, and expiration). Specifically:
      * Minimum length of 12 characters.
      * Require a mix of uppercase and lowercase letters, numbers, and symbols.
      * Password expiration of 90 days.
      * Prevent password reuse.
      * Use a password complexity library (e.g., zxcvbn) to enforce strong passwords.
    - Consider using Single Sign-On (SSO) integration with our existing identity provider (Azure AD) for a seamless user experience and centralized authentication management. Use protocols like OpenID Connect.
  + Session management:
    - Implement secure session management practices, including appropriate session timeouts, secure session IDs, and protection against session fixation and hijacking.
    - Use HTTP-only and Secure cookies to protect session IDs.
    - Implement session timeouts (e.g., 20 minutes of inactivity for regular users, 30 minutes for administrators).
    - Rotate session IDs after successful login.
    - Use a strong cryptographic random number generator for session ID generation.

**2. Infrastructure Security (Azure Hosting):**

* **High Availability (HA) and Redundancy):**
  + Design the solution architecture for high availability and fault tolerance within Azure.
  + Utilize Azure Availability Zones or Availability Sets to protect against hardware failures and datacenter outages. Deploy the web application and database across multiple availability zones.
  + Implement load balancing across multiple instances of the web application. Use Azure Load Balancer or Azure Application Gateway. Configure health probes to automatically remove unhealthy instances from the load balancer pool.
  + Database redundancy (e.g., Azure SQL Database with active geo-replication or availability zones). For Azure SQL, use Active Geo-Replication for disaster recovery, or Availability Zones for high availability within a region. For other databases (e.g., MySQL, PostgreSQL), use read replicas and clustering.
* **Web Application Firewall (WAF):**
  + Deploy a Web Application Firewall (WAF) in front of the website and e-shop to protect against common web application attacks.
  + Use Azure WAF with OWASP Top 10 protection, bot protection, and custom rules as needed. Configure the WAF to:
    - Block common attack patterns (e.g., SQL injection, XSS).
    - Filter out malicious bots.
    - Enforce allowed HTTP methods.
    - Implement rate limiting to prevent denial-of-service attacks.
  + Regularly update WAF rules to address new vulnerabilities. Automate WAF rule updates using a CI/CD pipeline.
* **Anti-DDoS Protection:**
  + Implement DDoS protection to mitigate denial-of-service attacks.
  + Use Azure DDoS Protection Standard for network layer (Layer 3/4) and application layer (Layer 7) defense. Configure:
    - Network-level DDoS protection on the Virtual Network.
    - Application-level DDoS protection via Azure WAF.
  + Configure alerting and monitoring for DDoS attacks. Integrate with Azure Monitor to get notified about DDoS attacks and traffic anomalies.
* **Azure Security Best Practices:**
  + Follow Azure security best practices, including:
    - Principle of Least Privilege for all Azure resources. Use Azure RBAC to grant only the necessary permissions to users, groups, and applications. Define custom roles if needed.
    - Network segmentation using Virtual Networks (VNets) and Network Security Groups (NSGs). Create separate VNets for the web application, database, and other components. Use NSGs to filter traffic between subnets and to/from the internet. Example NSG rules:
      * Allow HTTPS traffic (port 443) from the internet to the web application subnet.
      * Allow SQL traffic (port 1433 for SQL Server, 5432 for PostgreSQL) from the web application subnet to the database subnet.
      * Deny all other traffic by default.
    - Secure configuration of Azure services. Follow the Center for Internet Security (CIS) benchmarks for Azure.
    - Regular security assessments and penetration testing. Schedule penetration tests at least annually, and after any major application changes. Use a combination of automated scanning tools and manual testing.
* **Secure Key Management:**
  + Use Azure Key Vault to securely store and manage cryptographic keys, certificates, and secrets.
  + Control access to Key Vault using Azure RBAC. Grant access to Key Vault only to the applications and users that need it.
  + Rotate keys and certificates regularly. Automate key rotation. For example, rotate encryption keys every 90 days and SSL/TLS certificates annually.
* **Data Encryption:**
  + Encrypt data at rest:
    - Use Azure Storage Service Encryption for data stored in Azure Storage. Enable server-side encryption (SSE) for blob storage and file storage. Use customer-managed keys (CMK) for more control.
    - Use Transparent Data Encryption (TDE) for Azure SQL Database. Enable TDE to encrypt the database files. Use CMK for TDE as well.
  + Encrypt data in transit:
    - Enforce HTTPS for all website and e-shop traffic. Obtain a valid SSL/TLS certificate from a trusted Certificate Authority (CA).
    - Use TLS 1.2 or later. Disable older versions of TLS (e.g., TLS 1.0, TLS 1.1) as they are vulnerable.
    - Implement HTTP Strict Transport Security (HSTS). Configure HSTS on the web server to force browsers to always use HTTPS.

**3. Application Security:**

* **Secure Development Practices:**
  + Implement a Secure Software Development Lifecycle (SSDLC). Integrate security into every stage of the software development process, from requirements gathering to deployment and maintenance.
  + Conduct regular security code reviews. Use static analysis security testing (SAST) tools (e.g., SonarQube, Veracode) and manual code reviews to identify vulnerabilities.
  + Use static application security testing (SAST) and dynamic application security testing (DAST) tools. Integrate SAST and DAST tools into the CI/CD pipeline to automatically scan for vulnerabilities. DAST tools (e.g., OWASP ZAP, Burp Suite) should be used to test the running application.
  + Address vulnerabilities identified through the OWASP Top 10 and other relevant security standards. Prioritize remediation of high-risk vulnerabilities.
* **Input Validation and Output Encoding:**
  + Implement robust input validation on all user-supplied data to prevent injection attacks (e.g., SQL injection, cross-site scripting). Use a validation library (e.g., OWASP Validation Regex Repository) and server-side validation.
  + Use output encoding to prevent cross-site scripting (XSS) vulnerabilities. Use context-aware output encoding. For example, use HTML entity encoding for output in HTML, and JavaScript encoding for output in JavaScript. Frameworks like React, Angular, and Vue.js have built-in protection.
* **Vulnerability Management:**
  + Establish a process for identifying, assessing, and patching vulnerabilities in the CMS, e-shop platform, and any third-party libraries.
  + Conduct regular vulnerability scans. Use vulnerability scanning tools (e.g., Nessus, OpenVAS) to scan the web application, servers, and network infrastructure.
  + Subscribe to security advisories and promptly apply security patches. Automate patching using a configuration management tool (e.g., Ansible, Puppet).
  + **Vulnerability Patching and Remediation:** All identified vulnerabilities, whether from vulnerability assessments, penetration tests or secure code reviews, must be patched and remediated in a timely manner. This includes applying vendor-supplied patches, developing and deploying custom patches, or implementing other compensating controls. The remediation process should include:
    - Prioritizing vulnerabilities based on severity (e.g., Critical, High, Medium, Low).
    - Establishing Service Level Agreements (SLAs) for remediation. The following SLAs are proposed:
      * Critical: 7 days
      * High: 30 days
      * Medium/Low: 90 days, next release but not more than 6 months if no change is planned
    - Tracking and documenting the remediation status of all identified vulnerabilities.
    - Verifying the effectiveness of patches and remediations.
    - Communicating the status of vulnerabilities and remediation efforts to the beneficiary, including any significant delays or challenges.
* **API Security:**
  + Secure all APIs used for integration with existing apps.
  + Implement authentication and authorization for API access (e.g., OAuth 2.0, API keys). Use access tokens with short expiration times.
  + Use input validation and output encoding to prevent API-related attacks.
  + Rate limit API requests to prevent abuse. Implement rate limiting (e.g., using Azure API Management) to prevent denial-of-service attacks and brute-force attacks.
  + Document API security measures. Use the OpenAPI Specification (formerly Swagger) to document API endpoints, authentication, and authorization requirements.

**4. Logging, Monitoring, and Auditing:**

* **Security Logging:**
  + Implement comprehensive security logging for all relevant events, including:
    - Authentication and authorization attempts (successes and failures).
    - Access to sensitive data.
    - System errors and exceptions.
    - Security-related configuration changes.
    - Application logs (e.g., using Log4j, Serilog).
    - Web server logs (e.g., Apache or IIS access and error logs).
    - Database logs (e.g., SQL Server audit logs, PostgreSQL audit logs).
* **Log Retention:**
  + Retain security logs for a minimum of 3 months to support security investigations and compliance requirements. Store logs in a secure, centralized log management system (e.g., Azure Monitor Log Analytics).
* **Log Management and Analysis:**
  + Integrate with Azure Monitor and Application Insights for centralized log management, analysis, and alerting.
  + Configure alerts for suspicious activity, security incidents, and system anomalies. Use Azure Monitor to create alerts for:
    - Multiple failed login attempts.
    - Access to sensitive data by unauthorized users.
    - Changes to security-related configurations.
    - Unexpected traffic patterns.
  + Regularly review logs to identify potential security threats.
* **Reporting:**
  + Implement reporting capabilities for security-related events.
  + Generate reports on access control, authentication, errors, and other relevant security data.
  + Automate report generation where possible. Use a reporting tool to schedule and generate security reports.
* **Intrusion Detection/Prevention:**
  + Consider implementing an Intrusion Detection System (IDS) or Intrusion Prevention System (IPS) to detect and respond to malicious activity. Azure provides some of this functionality. Use:
    - Azure Security Center for threat detection.
    - Azure Network Watcher for network traffic analysis.
    - Host-based IDS/IPS on the web servers, if necessary.

**5. Data Backup and Recovery:**

* **Daily Backups:**
  + Implement daily backups of the website, e-shop data, and database. Use Azure Backup to automate backups.
* **Backup Retention:**
  + Retain backups for 3 months. Store backups in Azure with appropriate retention policies.
* **Secure Backup Storage:**
  + Store backups securely in Azure, with appropriate access controls.
  + Encrypt backups at rest. Use Azure Storage Service Encryption for backups.
* **Backup Testing:**
  + Regularly test backup and recovery procedures to ensure data can be restored successfully in a timely manner. Conduct regular disaster recovery drills (at least on an annual basis).
* **Disaster Recovery Planning:**
  + Develop a disaster recovery (DR) plan that outlines the steps to restore the website and e-shop in the event of a major outage.
  + The DR plan should include:
    - Identification of critical systems and data.
    - Recovery time objectives (RTOs) and recovery point objectives (RPOs). Define RTO and RPO for the website and e-shop (e.g., RTO of 4 hours, RPO of 1 hour).
    - Step-by-step recovery procedures.
    - Regular testing of the DR plan. Test the DR plan at least annually.

**6. Compliance:**

* **Regulatory Compliance:**
  + Ensure the solution complies with all relevant regulations, including:
    - GDPR: Implement data privacy measures, including data minimization, purpose limitation, and data subject rights.
    - Any other applicable industry-specific regulations.
* **Security Policies:**
  + Develop and implement comprehensive security policies and procedures that cover all aspects of the website and e-shop security. Policies should cover:
    - Access control
    - Password management
    - Data protection
    - Incident response
    - Vulnerability management
    - Backup and recovery

**Declaration**

We hereby confirm that the information provided in this annex is complete and accurate, and that, in the event the contract is awarded, we commit to fully complying with the security requirements established by RetuRO.

**[Authorized Representative Signature]** ………………………………………………………………………………………………………  
**[Full Name]** ……………………………………………………………………………………………………………..……………….…………………  
**[Position]** ……………………………………………………………………………………………………………..……………….…………………….  
**[Bidding Company]** …………………………………………………………………………………………………………..……………….………..  
**[Date]** ……………………………………………………………………………………………………………..……………….…………………………