Application and Program Development Guidelines and Procedures

<table>
<thead>
<tr>
<th>Version</th>
<th>Approval Date</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>February 27, 2022</td>
<td>Chief Technology Officer</td>
</tr>
</tbody>
</table>

I. Purpose of Procedure

The purpose of this procedure is to provide requirements and guidelines for the development and maintenance of applications and programs.

II. Procedure Scope

This procedure covers applications and programs developed or revised by all employees, interns, contractors, and third parties who may perform programming.

III. Definitions

For a complete list of definitions, refer to the Glossary.

IV. Procedures

This procedure shall be reviewed periodically for any potential advancement in best practices or impact of any environmental changes.

Development shall follow HSX’s formal System Development Lifecycle (SDLC), which covers request initiation and authorization, requirements definition, analysis, communication, conflict detection and resolution, and evolution of requirements as may be required.

Code

- Input Validation
- User defined inputs such as query string parameters, request bodies, request headers, form fields, etc. should be validated and sanitized. Checks can include but not are not limited to, file type, text format, data structure, and size.

  - **State**
    - Avoid storing values in global state. This can lead to unpredictable results.
    - Functions that take input and always deliver the same output should be favored over methods that take input and return different output depending on internal state. This makes code easier to reason about, more predictable, and more testable.

  - **Errors**
    - Errors should be clear and useful. They should help a developer effectively find the relevant issue.
    - Use standard error classes and codes where possible. Only create custom errors if standard errors do not accurately describe the failure case. However, error messages can (and often should) be custom and should specifically describe the error case.
    - Capturing of errors should be centralized in a service where possible, such that errors can be handled, transformed, and logged in a consistent manner.
    - End users should receive a descriptive error summarized to a level that helps them fully understand what they can or should do differently to rectify the error. Note that logical code errors should always result in an "UnexpectedError" or "InternalServerError". Low level details of logical code errors should never be revealed to end users, however, they should be descriptively logged for consumption by developers. Examples of proper and improper end user error messages are offered below:

<table>
<thead>
<tr>
<th>Type of Error</th>
<th>Proper End User Error Message</th>
<th>Improper End user Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>User supplied a phone number in an invalid format</td>
<td>Phone number format invalid</td>
<td>Phone number could not be parsed</td>
</tr>
<tr>
<td>Required phone number field was left empty</td>
<td>Phone number is required</td>
<td>Field cannot be null</td>
</tr>
<tr>
<td>Invalid URL was used</td>
<td>Either “Not Found” or &quot;API not found”</td>
<td>Request invalid</td>
</tr>
<tr>
<td>Logical code error occurred</td>
<td>Internal Server Error</td>
<td>Failed to require library 'lodash'</td>
</tr>
</tbody>
</table>

- **Comments**
Comments in code should be clearly written and useful.

- Comments should give high level explanations of more complex code. If a large body of code feels like it requires a large number of comments, consider whether the code might be unnecessarily complex.
- Remove old commented code unless it is highly likely to be re-enabled or used for valuable context in the future.
- Comments specifying future necessary or desired work should start with “TODO”.
- In the event that HSX develops source code, access to program source code and associated items (such as designs, specifications, verification plans and validation plans) will be strictly controlled and protected, in order to prevent the introduction of unauthorized functionality and to avoid unintentional changes. When doing system development (e.g., applications, databases), output validation, it must be either manually or automatically performed and must include:
  - plausibility checks to test whether the output data is reasonable;
  - reconciliation control counts to ensure processing of all data;
  - providing sufficient information for QA resources to determine the accuracy, completeness, precision, and classification of the information;
  - procedures for responding to output validation tests;
  - defining the responsibilities of all personnel involved in the data output process;
  - and documenting the activities of the data output validation process. In the change management ticket.

**Source Control and Versioning**

- All code and system configurations should be stored in source control and versioned where possible.
- A git-based source control system should be used (currently Bitbucket)

**Authentication/Authorization**

- All user accessible endpoints and access points to applications that expose any sensitive data (whether PII or otherwise) should be authenticated and authorized using industry best practices. Whenever possible use existing libraries and third-party services to implement authentication and authorization. Authentication and authorization logic should not be custom-built unless there is a strong need for the given use case.

**Configuration**
• Services and applications should be environment independent so that they can easily be
ported and promoted through development, testing, staging, and production. Differences in
the implementations between these environments should be injected via environment
variables at build-time (when system artifacts are built) or run-time (when the services
launch). Migration of software and configuration changes from development to operational
status shall not occur until successful tests are completed based upon pre-defined test scripts
and have been authorized through HSX change management procedures.
• Automated controls shall be incorporated in the information system, supplemented by manual
cpyontrols as needed throughout the SDLC.

Secrets

• Secrets such as passwords and private keys should never be stored in source code.
• Secrets should be injected at runtime via environment variables or through configuration
management systems that store the secrets in an encrypted form protected behind
authentication and authorization systems.
• Secrets should not be output in system logs and messages unless absolutely necessary. If
secrets are logged, any such logs sent to centralized logging systems should be removed
before ingestion into the logging system.

Dependencies

• Third party software libraries and systems must be evaluated before being selected to
be used as dependencies for developed services and applications. Factors to be
evaluated should include, but are not limited to:
  o frequency of use in the open source and enterprise software communities,
  o frequency of code maintenance,
  o number of maintainers,
  o reputation of maintainer(s),
  o acceptance into foundations such as Apache and the Cloud Native Computing
    Foundation, and
  o quality of source code and test coverage (if available)
  o results of scanning for security vulnerabilities
• HSX has established restrictions on the use of open-source software which must be legally
licensed, adhere to secure configuration policies and be authorized by the HSX CTO/CISO or
Director of IT through the change management procedures.
Logging and Monitoring

- Logs and metrics from services and applications should be sent to central aggregation systems where logs and metrics can be organized, collated, and leveraged for monitoring and alerting. Logs and metrics should contain information to help identify which endpoint they came from (server, container, function, etc.), which environment they correspond to (dev, test, prod, etc.), the creation timestamp, their severity, and any other information that could help a developer identify issues or patterns.

Network

- Network services should be exposed over the least possible interfaces, protocols, and ports. Services, especially ones that expose sensitive data such as PII or secrets, should be accessible only from consumers of these services.
  - For example, a MongoDB database server listening on port 27017 that contains PII should be only open on ports 27017 and 22 for SSH and only from IPs or security groups corresponding to the services that consume that MongoDB database. That server should not expose ports 443, 80, 8080, etc., unless necessary. Other services (such as a consuming service's load balancer or an entirely different service) that do not need to consume that database should not have any network access to that database server.

Encryption

- All data should be encrypted in transit and at rest.
- Always favor HTTPS over HTTP endpoints. HTTP endpoints should redirect to HTTPS.
- All disks used by services should be encrypted.
- Use the latest encryption protocols and algorithms that are supported by the given use case.

Security Evaluation and Testing

- Publicly exposed services, particularly ones that accept user input, should be evaluated for vulnerabilities regularly or before first deployment to production. Evaluation can be performed by third-party services, third party tools, or in-house tools.
<table>
<thead>
<tr>
<th>Responsible Owner:</th>
<th>Chief Technology Officer</th>
<th>Contact: email</th>
<th><a href="mailto:Brian.Wells@healthshareexchange.org">Brian.Wells@healthshareexchange.org</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Approved By:</td>
<td>Brian Wells</td>
<td>Version #</td>
<td>1.1</td>
</tr>
<tr>
<td>Current Approval Date:</td>
<td>July 1, 2019</td>
<td>Last Reviewed</td>
<td>October 26, 2023</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>December 6, 2022</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>February 6, 2022</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>October 2, 2021</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>October 17, 2020</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>December 30, 2019</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Date Procedure to go into Effect:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>July 1, 2019</td>
</tr>
<tr>
<td>Related Documents:</td>
<td>Source Code Management Policy</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>