Application Development Standards

Contents

1. Revision History

Version	Author	Published
2024-08	Ryan J. Johnson	2024
2026-04	Ryan J. Johnson	2026

2. Introduction

2.1 Purpose

These standards serve to ensure the maintainability, security, integrity, availability, and uniformity of the Agency's IT infrastructure and software portfolio. These general standards are not intended to be specific to any particular Development Language, Framework, Component, or Service, or combination thereof. See **Appendix B: Acceptable Languages and Frameworks** for detailed requirements or to request permission to use a non-listed technology.

2.2 Scope

This document defines the standards to which all software developed by, for, or on behalf of the Agency must adhere. This includes software developed by Agency employees, contractors, vendors, and any other entity when such software is to be utilized by or integrated with any Agency system, person, or process.

Unless otherwise indicated, all standards defined herein apply to software without limitation as to where or by what mechanism the software is hosted or deployed.

2.3 Enforcement and Compliance

These standards must be reviewed by the responsible development team prior to the commencement of any architectural design or software development work to ensure that planned work aligns with the standards and to allow for sufficient time for the review of any questions or requested variances.

At each code review during the development of any software, these standards must be consulted and applied to the work product by a designated reviewer. Any deviations identified during this review must either be remediated or documented as a permitted variance from these standards by the Agency.

3. General Standards

3.1 Privileged, Sensitive, and Dynamic Information

Credentials, sensitive or privileged information, and data that is subject to frequent change shall not be stored in the Source. This information includes, but is not limited to:

1. Usernames
2. Passwords
3. Private or public keys
4. Certificates
5. Tokens
6. Client Identifiers
7. E-mail addresses
8. Server addresses (hostnames, IP addresses, aliases)
9. File or directory paths
10. Phone numbers
11. Individual or business names

If such information is required for the operation of the Application, the Agency will provide a means by which the information may be obtained from a secure location by the Application at startup and/or at runtime.

3.2 Prohibited Components and Frameworks

No Application shall make use of any Third-Party Component or Framework which has reached or passed its End of Life or Support as declared by its author, will reach its End of Life or Support as declared by its author within two (2) years of its time of inclusion in the Application, or has become defunct by absence of maintenance by its author within six (6) months of its time of inclusion in the Application. Maintenance may include responding to issue reports, communications of intent to release updates to the product, or updates to the product.

Without prior approval from the Agency, no Application shall make use of any Third-Party Component, Framework, or Service that:

- Has not reached Maturity, or
- If published under a commercial license, said license has not been properly acquired by or, if permitted by the terms of the license, transferred to the Agency.

3.3 Prohibited Practices

3.3.1 Direct Inclusion of Components

Components must be included in an Application via a dependency management framework. Exceptions may be granted upon written request.

3.3.2 Modification of Third-Party Components

Unless written permission is granted by the Agency, and such modification does not violate the terms of the Component's license, no Third-Party Component shall be modified in any way. Extending a Component without modifying the original Component via methods such as inheritance and composition is permitted.

3.3.3 Exposure of Infrastructure

An Application shall not expose system properties, environment variables, directory structures, or other infrastructure-related information to the user.

3.3.4 Intellectual Property Rights

No Application Source shall be permitted to bear the name of the vendor(s) responsible for its development in filenames, directory names, package names, class names, project names, user interfaces, API responses, or in any other place that may imply the vendor's retention of ownership or intellectual property rights without prior approval by the Agency.

4. Third-Party Components

The usage of any Third-Party Components shall require approval by the Agency prior to the deployment of the Application to any Agency service, server, or device, and prior to the connection of the Application to any Agency service, server, or device.

At no time shall any Component licensed under the GNU Affero General Public License (AGPL) or equivalent "copyleft" licenses be used in any Application.

It is understood that the usage of non-free Components may occasionally be necessary. The usage of non-free Components shall require approval by the Agency prior to their inclusion in the Application at any point in the Application's development.

A full manifest of all Third-Party Components shall be provided to the Agency for review upon request. The manifest shall include:

1. The common name of the Component
2. The version of the Component
3. The author of the Component
4. The name and version (where applicable) of the license under which the Component is distributed for the version of the component provided in the manifest
5. The inclusion mechanism of the Component (e.g., Maven, NPM, Composer, etc.)

5. Third-Party Services

No Application shall make any connections to any Third-Party Service by any mechanism without prior approval by the Agency.

6. Vulnerabilities in Third-Party Components

All Third-Party Components, Frameworks, and Services incorporated into an Application shall be evaluated for known security vulnerabilities both prior to their initial inclusion and on a continuous basis throughout the development of the Application. The Agency requires that all vendors and development

teams exercise due diligence in identifying, assessing, and remediating vulnerabilities in a timely and responsible manner.

6.1 Vulnerability Identification and Assessment

All Components shall be scanned using industry-recognized vulnerability detection tools and shall be reviewed against reputable vulnerability databases. The assessment shall include, at minimum:

- The Common Vulnerabilities and Exposures (CVE) identifiers associated with the Component.
- The Common Vulnerability Scoring System (CVSS) Base Score and associated severity category.
- The applicability of each vulnerability to the Component as configured and used within the Application.
- The availability and feasibility of vendor-issued patches, upgrades, or recommended mitigations.
- Any compensating controls implemented within the Application or its environment.

6.2 Inclusion and Remediation Requirements

The following requirements apply to all Components, Frameworks, and Services:

- Critical Vulnerabilities (CVSS 9.0–10.0)
  Components with one or more Critical vulnerabilities shall not be included in any Application. If a Critical vulnerability is discovered in an already-included Component, the Component shall be removed, disabled, or replaced immediately, unless the Agency authorizes a temporary exception under documented emergency conditions.
- High Vulnerabilities (CVSS 7.0–8.9)
  Components containing High vulnerabilities may be included or retained only after the vendor submits formal justification, including a technical impact analysis, a description of compensating controls, and a clear plan and timeline for remediation. Prior written approval from the Agency is required.
- Medium and Low Vulnerabilities (CVSS 0.1–6.9)
  Components containing Medium or Low vulnerabilities may be included if the vulnerability does not materially impact the Application's confidentiality, integrity, or availability and can be demonstrated to not impact other Agency resources; or if effective mitigations are in place. These vulnerabilities remain subject to ongoing monitoring and must be addressed as updates become available.

6.3 Ongoing Monitoring and Notification

Vendors and development teams shall continuously monitor all Components for newly disclosed vulnerabilities throughout the development of the Application and for the life of the Application throughout its maintenance phase. The Agency's Information Security Manager (ISM), the OTIS Coordinator, Contract Manager, and the Enterprise Architect shall be notified immediately upon the identification of any new High or Critical vulnerability affecting a Component in use. Upon notification, the Agency may require remediation, mitigation, or replacement of the Component.

6.4 Exception Process

Exceptions to these requirements may be granted only in circumstances where no feasible alternative Component exists and where the removal of the Component would impair the Application's

functionality. Any exception must be formally requested, documented, and approved by the Information Security Manager and shall include:

- A justification for continued use
- A mitigation strategy
- A remediation or replacement timeline
- A designated point of responsibility

All exceptions remain subject to periodic review and may be revoked at any time.

See **Appendix A: Vulnerability Assessment** to report vulnerabilities and to request exceptions.

7. Practices, Style, and Conventions

7.1 Naming

7.1.1 Applications

Applications that are to be deployed within the Agency's infrastructure must be named following these rules, provided that the naming conventions are compatible with the technologies and deployment environment:

1. The Application's name must be in "lower-hyphen-case."
2. The first segment of the name must be the common initialism or abbreviation of the division, office, or other organizational unit within the Agency that is the primary sponsor of the Application's development.
3. All segments of the name must be separated by hyphens. Each word, initialism, or other portion of an Application's name must be its own segment.
4. If an Application consists of multiple modules that are packaged into separate deployment units, such as a frontend and backend, the name of each unit must indicate its purpose, excluding the frontend, which must be the name of the Application.
5. The first segment of the names of Applications developed for enterprise use in support of other divisions must be "dep," such as "dep-account-service."

Acceptable Examples

- orcp-sva
    - "orcp": The common initialism of the Office of Resilience and Coastal Protection (ORCP).
    - "sva": The name of the application.
    - This is the frontend (user interface) of the "SVA" (Statewide Vulnerability Assessment) application.
- orcp-sva-api
    - "api": Indicates that this is the backend of the "orcp-sva" unit.
- dwm-swift
    - "dwm": Division of Waste Management (DWM).
    - "swift": Solid Waste Information Field Tracking.

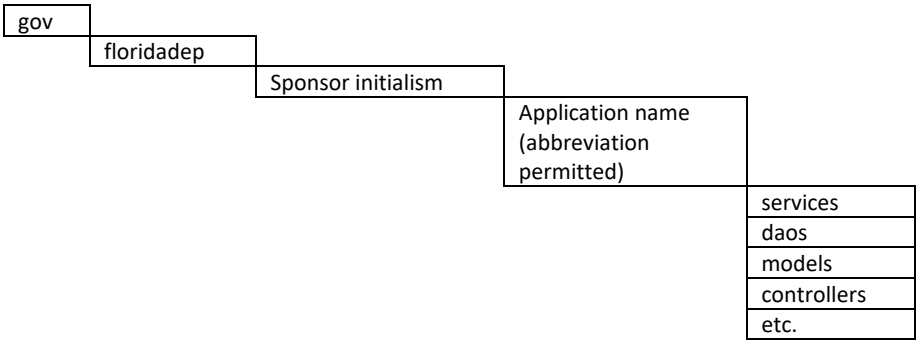
Unacceptable Examples

- dwrm-waterservice

    - "water" and "service" must be separated by a hyphen into "water-service," forming the acceptable name "dwrm-water-service."
- land-tracker
    - No indication is given of sponsorship.
- DslLandTracker
    - Contains uppercase characters.
    - Segments are not separated by hyphens.

7.1.2 Packages and Directories

For applications written in languages or frameworks that make use of namespaces, packages, or other hierarchical grouping methodologies (including directory structures), the hierarchy of names must follow this convention:



A fully qualified class name (i.e., including its package name or namespace) for a:

1. Service class named "UserService"
2. In an application named "baz"
3. Developed for the Office of Technology and Information Services (OTIS)

is:

gov.floridadep.otis.baz.services.UserService

7.1.3 Variables, Constants, Files, Classes ("Elements")

1. All Elements of the Application Source shall be named in a descriptive manner.
2. Abbreviation of words in these named items is only acceptable when necessary to satisfy compiler limitations or language syntax rules.
3. All Element names must be correctly spelled American English words, or phrases thereof, from the Oxford English Dictionary except when the name is an established initialism, abbreviation, proper noun, trademark, or service mark.
4. Elements shall be named as singular nouns or verbs except when they represent a collection of other elements or operators.

7.1.4 Naming Variances

If the naming conventions described in the preceding sections are incompatible with the technologies used, are otherwise demonstrably impractical, or are in contravention with the standards for the technologies used in the Application's development, written approval must be obtained from the Agency to deviate from these conventions.

7.2 Practices and Architectural Requirements

7.2.1 API Design

When RESTful APIs are implemented, the following requirements apply:

- All endpoints must use plural nouns, following "lower-hyphen-case." The HTTP verb describes the operation; the URI describes the subject. Correct example: /v1/registered-users/doe_j. Incorrect: /getRegisteredUsers/doe_j.
- APIs must be versioned (e.g. /v1/my-endpoint).
- Endpoints that may reveal sensitive information and any endpoints that may modify data must be secured. An industry standard means of securing API endpoints will be provided by the Agency when necessary.
- API responses must follow a predictable pattern, such as the Hypertext Application Language (HAL) JSON convention. Excluding binary or empty (header-only) responses, all API responses from all API endpoints within an Application must follow the same convention.
- Unless otherwise necessary, API response payloads must be JSON.
- Pagination is required when an indeterminate number of records may be returned or when the response payload may exceed 1,000 records.
- Any client-side validation must also be performed by the API endpoint.
- Only standard HTTP response codes may be used. The response code should meaningfully reflect the outcome of the operation.
- No stack traces, implementation details, internal identifiers, or infrastructure information may be included in the output of an API endpoint.

API technologies other than REST may be permitted at the Agency's sole discretion.

7.2.2 Logging

A standardized logging system must be implemented in the Application such that all log messages emitted by the Application are emitted to the console (stdout/"standard out" and stderr/"standard error").

Log messages must not contain sensitive information, such as passwords, tokens, information exempted from public record by Florida Statute 119.071, or other legally protected confidential information.

When possible, logs messages must contain a correlation ID associating the message with the user or service responsible for the generation of the log message.

Log messages must be written with an appropriate severity level so that standard tracing/debugging messages do not flood or obscure messages with a higher severity level. Errors must only be logged once, unless additional contextual information is added with subsequent messages. "Error" severity messages must only be written to indicate an event that requires investigation has occurred.

7.2.3 Error Handling

Errors and exceptions must never be silently ignored. All exceptions must be caught, logged, and addressed in such a way that the user or API consumer is given a clear indication that a problem occurred and guidance as to how to proceed.

7.3 Other Conventions

In addition to the standards defined herein, all Application code shall adhere to the appropriate style guide listed below. For Application code written in a programming, scripting, or markup language that is not identified below, the vendor shall request a style guide from the Agency.

Language	Style Guide	Reference
Java	Google Style Guide	https://google.github.io/styleguide/javaguide.html
PHP	WordPress PHP Coding Standards	https://developer.wordpress.org/coding-standards/wordpress-coding-standards/php/
HTML and CSS	Google HTML/CSS Style Guide	https://google.github.io/styleguide/htmlcssguide.html
JavaScript	Google JavaScript Style Guide	https://google.github.io/styleguide/jsguide.html
TypeScript	Google TypeScript Style Guide	https://google.github.io/styleguide/tsguide.html

8. Definitions

Term	Definition
Application	The entirety of packaged module(s) that may be executed or deployed
Agency	The Florida Department of Environmental Protection
Component	Any work that may be included in another work to enable or enhance its functionality
Development Language	Any scripting, programming, imperative or declarative configuration, or other language intended for execution or interpretation by a computer system
End of Life or End of Support	The date at which a product is no longer maintained by its primary maintainer
Framework	Any set of Components that, together, provide a means of constructing an Application or larger/composite Component
Maturity	As it applies to Components, Frameworks, and Services: having had at least one (1) viable release that is not labeled as 'Beta,' 'Test,' 'Alpha,' 'Release Candidate (RC),' or any other marker to indicate that it is not stable
Service	Any discrete source or consumer of data with which another Service or Application may communicate
Source	The entirety of the uncompiled code which, once compiled and packaged, constitutes the Application
Third-Party Component	Any Component produced by an entity other than the Agency, or Component that transitively includes such a Component
Third-Party Framework	Any Framework produced by an entity other than the Agency
Third-Party Service	Any Service provided by an entity other than the Agency

Appendix A: Vulnerability Assessment

Project or application name: _____

Date of assessment: _____

Tool(s) used for assessment: _____

Assessor's name and contact information: _____

Date reviewed by Information Security Manager: _____

Following the example provided, list all identified vulnerabilities.

CVE Identifier	CVSS Base Score / Severity	Applicability to Component	Patch / Mitigation Strategy and Timeline	Compensating Controls	Exception Requested/Approved
Example: CVE-2021-44228	10.0 / Critical	Vulnerability is applicable when the Log4j library is used with default JNDI lookup enabled.	Vendor patch available; upgrade to Log4j 2.17.0 or later. This will be patched immediately.	JNDI lookups disabled at runtime as temporary mitigation.	Requested: No Approved: N/A
					Requested: Approved:

Appendix B: Acceptable Languages and Frameworks

Purpose

Only the Languages, Frameworks, and versions and variations thereof enumerated in this document are permitted for use in new development. These are only the core Languages and Frameworks, it is understood that extensions to the listed technologies may be used, provided that such extensions meet the requirements defined in the Application Development Standards.

Scope

These Languages and Frameworks will be compiled into the Application's container image by the Agency's CI/CD Pipelines. The runtime environment is provided by the Agency and no changes to the environment will be permitted.

Languages

Java

All new Java Applications must use Java 17. No usage of features that are unique or specific to any vendor's release of the Java Virtual Machine (JVM) is permitted; only features which exist in stable, standard releases of Java 17 may be used. No experimental features may be used.

The usage of additional JVM arguments is strongly discouraged and requires approval from the Agency.

All Java 17 Applications will be deployed in a single-tenant instance of Apache Tomcat 10.

All Java Applications must use Apache Maven 3.8.5 or newer for dependency management.

PHP

All PHP Applications must be written in PHP 8.2. PHP will be executed with the FPM engine.

All PHP Applications must use Composer 2.5 for dependency management.

Frameworks

For the purpose of this document, a "Framework" is defined as a structured collection of Components that establishes the foundational architecture of an Application and provides standardized mechanisms for configuring, initializing, and executing the Application at runtime. Frameworks typically provide features such as inversion of control, dependency injection, routing, request and response handling, error handling, configuration management, and other core application lifecycle functions that are not supplied by the underlying runtime environment.

The following Frameworks are the only core application Frameworks permitted for new development:

Framework	Version	Notes
Spring Boot	3.x	The latest stable version of Spring Boot 3 and its accompanying modules may be used
Laravel	11.x	The latest stable version of Laravel 11 for PHP 8.2
Thymeleaf	Matches Spring Boot	The version of Thymeleaf that corresponds to or is officially supported by the version of Spring Boot in use.
React (TypeScript)	18.x	The latest stable version of React 18
Next.js (TypeScript)	15.x	The latest stable version of Next.js 15
Angular (TypeScript)	21	The latest stable version of Angular 21

Usage of a Framework that is not listed above and is not an extension to one of the Frameworks requires written approval from the Agency.