Admitto Unum

Admito Unum Software

1.0 Design Principles

`/1.0 Design Principles`

1.1 Expressive Design

`/1.0 Design Principles/1.1 Expressive Design`

Expresive Programing Design

References:

1. Domain-Driven Design (DDD)
2. The Concept of Domain-Driven Design Explained

1.2 Internal DSL Principles

`/1.0 Design Principles/1.2 Internal DSL Principles`

Internal DSL Principles

Using Domain Specific Language principles when desigin the libraries makes the API easier to use and understand. Admitto Unum projects will use DSL written in Java when do so make sense. In most cases the DSL is layered on top of lower implementations.

References:

1. Building an Intuitive DSL in Java
2. Creating Internal DSLs in Java & Java 8

1.3 Clean Code

`/1.0 Design Principles/1.3 Clean Code`

Clean Code

References:

1. Clean Coding in Java
2. Clean Code Explained – A Practical Introduction to Clean Coding for Beginners

1.4 Design Patterns

`/1.0 Design Principles/1.4 Design Patterns`

Design Patterns

DESIGN PATTERNS Creational Design Patterns Structural Design Patterns Behavioral Design Patterns

2.0 Conventions

`/2.0 Conventions`

2.1 Coding Conventions

`/2.0 Conventions/2.1 Coding Conventions`

Java Code Conventions

Introduction

The coding standards will following the Check Style and SonarLint conventions with some modifications describe here.

The maven build plugin **route66-code-quality** is used to enforce thse standards. Any conflict between checkstyle and SonarLint will be resolved by accepting SonarLint rule and the checkstyle rule will be modified to match the SonarLint rule.

Coding Standards

All projects and artifacts must adhere to Clean Code principles, built on SOLID OOP principles and Design Patterns. Design and build tools will be used to ensure designs and code adhere to standards (See Development Environment).

File Organization

Class files will be organized as described in 3 - File Organization.

Class declarations should generally follow 11.1 Java Source File Example and organize members and fields by the following scope order as specified in the JVM Specification.

1. public
2. protected
3. package
4. private

Indention

Two spaces should be used as the unit of indentation.

Line Length

Avoid lines longer than 120 characters.

Comments and Documentation

Documentation comments generally follow the How to Write Doc Comments for the Javadoc Tool.

Comments should be written in complete sentences and should *not state the obvious.* For example, documentation for the method **getImage()** should not state *this method will get the image*. That fact is stated in method name. Instead comments should provide informative and useful information about the method. If the method's purpose is obvious by it's name then the method need not be commented. This may be the case in Jave Bean classes where setters and getters do nothing more the pass object references.

See Requirements for Writing Java API Specifications description and examples of each of the following comment types:

What should be documented

Classes are required to have accompanying documentation

- *public* methods should be documented
- *protected* methods should be documented
- *private* methods do not need to be comment since private methods in JavaDoc are not generated

Package Documentation

```
/**
 * { Executive summary: brief description of the purpose.}
 *
 * { OS/Hardware Dependencies. }
 *
 * { External specifications }
 *
 */
```

Class Documentation

```
/**
 * { Executive summary: brief one line description of the purpose of the class}
 *
 * { State Information:}
 *
 * { OS/Hardware Dependencies: }
 *
```

```
 * { Allowed Implementation Variances: }
 *
 * { Security Constraints: }
 *
 * { Serialized Form: }
 *
 * { External Secifications : }
 *
 * { Example of using the class }
 *
 * @author { author name }
 *
 * @since { version of the library the class was intruduced }
 */
```

Field Documentation

```
/**
 * { What this field models }
 *
 * { Range of valid values }
 *
 * { Null Value }
 *
 * @since { version of the library the method was intruduced }
 */
```

Method Documentation

```
/**
 * { Expected Behavior: }
 *
 * { State Transitions: }
 *
 * { Null Argument Values: }
 *
 * { Range of Return Values: }
 *
 * { Algorithms Defined: }
 *
 * { OS/Hardware Dependencies: }
 *
 * { Allowed Implementation Variances: }
 *
 * { Cause of Exceptions: }
 *
 * { Security Constraints: }
 *
 * { Example of using the class }
 *
 * @since { version of the library the method was intruduced }
 */
```

Naming Conventions

Naming conventions will follow the checkstyle Java Naming Conventions with the exeptions:

All package names will begin with "**admitto.unum.**" followed by the project and module id's. For example the route66 library Maybe interface would be in the **admitto.unum.route66.expressive** package.

For method, field and variable names see **1.3 Method Naming Conventions** for list and dictionary of standarized names.

References

- https://google.github.io/styleguide/javaguide.html
- http://www.oracle.com/technetwork/java/codeconventions-150003.pdf

2.2 Development Environment

`/2.0 Conventions/2.2 Development Environment`

1.2.1 Build Tools

`/2.0 Conventions/2.2 Development Environment/1.2.1 Build Tools`

Required Build Tools

1. Maven
2. Eclipse
3. Visual Code
4. GitHub
5. Jenkins CI
6. Admitto Unum Artifact Repository

Static Code Analysis

This project will utitlize the following automated analysis tools:

1. Code Formatting - formatting java source code using the Eclipse code formatter
2. CheckStyle - automates the process of checking Java code
3. SonarLint and SonarQube - identifies and helps you fix quality and security issues
4. OWASP Dependency Checking - attempts to detect publicly disclosed vulnerabilities contained within a project's dependencies

Maven Plugins

All Java artifacts will implement the following Maven build plugins and reporting.

1. Code Formatting.

Use the maven build plugin formatter-maven-plugin configured to use **JavaLambdaFormat** style. This style configuration is provied by the **admitone:code-quality-config** resource plugin.

Standard setup for formatting code is

```
    <!-- Automatically reformat source code to Java Lambda Format -->
    <plugin>
        <groupId>net.revelc.code.formatter</groupId>
        <artifactId>formatter-maven-plugin</artifactId>
        <version>2.18.0</version>
        <executions>
```

```
            <execution>
                <goals>
                    <goal>format</goal>
                </goals>
                <configuration>
                    <configFile>eclipse/JavaLambdaFormat.xml</configFile>
                </configuration>
            </execution>
        </executions>
        <configuration>
            <encoding>UTF-8</encoding>
            <excludes>
                <exclude>**/*Test.java</exclude>
                <exclude>**/*_Spec.java</exclude>
            </excludes>
        </configuration>
        <dependencies>
            <dependency>
                <groupId>admitone</groupId>
                <artifactId>code-quality-config</artifactId>
                <version>LATEST</version>
            </dependency>
        </dependencies>
    </plugin>
```

2. Setting up Checkstyle

Checkstyle will help educate and enforce our coding standards. You can set up your IDE to use Checkstyle to examine code for conformance to the standards. Learn more about the checks or Google the error message to find out why it complains about certain things.

```
    <!-- Verify code matches the AdmitOne Rout66 Java Lambda Function Requirements-->
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-checkstyle-plugin</artifactId>
        <version>3.1.2</version>
        <configuration>
            <configLocation>checkstyle.xml</configLocation>
        </configuration>
        <dependencies>
            <dependency>
                <groupId>com.puppycrawl.tools</groupId>
                <artifactId>checkstyle</artifactId>
                <version>10.0</version>
            </dependency>

            <dependency>
                <groupId>admitone</groupId>
                <artifactId>code-quality-config</artifactId>
                <version>LATEST</version>
            </dependency>
        </dependencies>
    </plugin>
```

3. SonarLint and SonarQube with SonarSannar

Execute the SonarLint and SonarQube analysis via a regular Maven SonarScanner.

TODO: add setup here

4. OWASP Dependency Checking

*TODO_*: add setup here

OWASP Dependency-Check

Eclipse

TODO

Jenkins

URL - http://admitone.ci:50080/jenkins/

AdmitOne Repository

TODO:

URL - http://admitone.repo:52180/

1.2.2 Documentation Tools

`/2.0 Conventions/2.2 Development Environment/1.2.2 Documentation Tools`

Markdown

Markdown for GitHub used to create all textual documentation.

C4 Model

C4 model is a lean graphical notation technique for modelling the architecture of software systems. It is based on a structural decomposition of a system into containers and components and relies on existing modelling techniques such as the Unified Modelling Language (UML) or Entity Relation Diagrams (ERD) for the more detailed decomposition of the architectural building blocks. (C4 model, Wikipedia)

Description

TODO: give description here.

Design Documenation

Design documentation will be created using C4builder. This documenation tool supports Markdown documents and C4 Models using C4-PlantUML. This tool is integrated with VSCode with *PlantUML v2.17.3* plugn.

The C4Builder tool generates a Web site locally for view using at (http://localhost:3000) while developing documentation.

When editing design documentation the C4Builder can be run to automatically regenerate the site content with

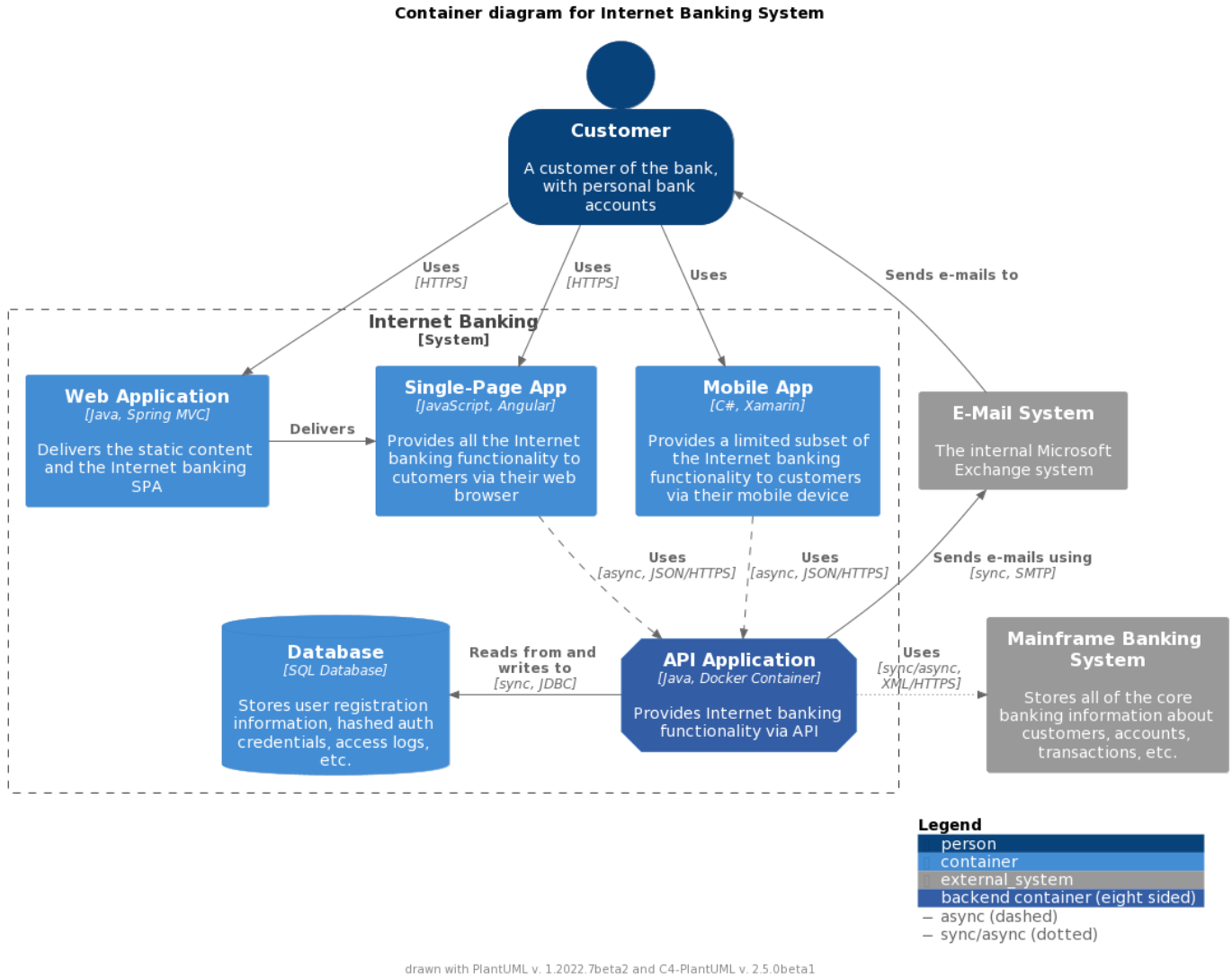
```
    c4builder site --watch
```

To run the Web site locally

```
c4builder site
```

C4-PlantUML

C4-PlantUML includes macros and stereotypes for creating C4 diagrams with PlantUML.



2.3 Method Naming Conventions

`/2.0 Conventions/2.3 Method Naming Conventions`

Overview

The following coding standards were collected from various technologies, frameworks and articles that form a collection of best practices for method names and what the purpose and/or implementation is expected.

Each of the identifiers presented by this convention form clauses used in whole or as part of an identifier. For example, 'as' can be used a a stand alone function call or part of a method name:

```
// Method that converts parameter and returns integer
Integer i = r.as(Integer.class);
```

```
    // Method that returns a value of a integer type.
    Integer i = r.asInteger();
```

Glossary Of Identifiers

As

Return a representation of an instance as a different type.

This clause identifies a method that converts the object to a different type.

Example

```
    var exam1 = Mabye.of(userRepo.findUserById(userId));
    var user = exam1.as(UserDetail.class).getName(); // Cast exam1 to a UserDetail and
get the users name.
    var str = userId.asString() // Return id value as a string.
```

Find

Return a subset of items matching criteria.

Method searches a collection or database for a specific instance(s) of a item.
The return type should be a Collection, Maybe or Optional.

The *find* clause should follow the Spring Data naming convention. See Spring Data .

Example

Prototype	Description
List findAll()	Return all instance of User
Optional findUserById(Long id)	Find a specific user with the given id
List findUserLikeLastName(String name)	Find all users with last name like name
List findUserByLastNameAndFirstName(..)	Find all users with last name and first name

From

Method that accept a reference to an Object as a parameter and returns a class with a value extracted from object.

Example

```
    var comp = Maybe.from(anOptional); // Creates a Maybe instance from the value of
Optional.
```

Get

Getter method to retrieve the value of a property.

The implementation of a getter should be simple and short; preferably written to qualify for Method Inlining in the JVM.

Of

Method that accepts a reference to an Object as a parameter and returns a wrapper class for the Object.

Example

```
    var response = Maybe.of(resp); // Creates a Maybe instance containing the value of
resp.
```

When

If the predicate is true then return the reference to this, otherwise, return a default instance.

Bisects method chaining depending on the result of a Predicate lambda function.
The return value will be one of two possible values depending on the *truth* returned by the Predicate. The when clause is frequently coupled with the *then* clause.

Example

```
    var someInt = Maybe.of(50); // Create a Maybe<Integer> with the value 50
    var result1 = someInt.when(p -> p > 25).then(p -> p * 10); // result1 will be 500
    var result2 = someInt.when(p -> p <=25).then(p -> p / 10); // result1 will be
None<Integer>
```

With

The *with* clause is used to make a call to a lambda that is a {@link Consumer} instance that will receive an instance of an object that the lambda function is expected to build. If the method name is *with* as a stand alone function then the object passed is usually the instance of the parent class.

This clause forms the DSL semantics is "with(x) do (y)".

Via

The *via* clause is used to make a call to a lambda that is a {@link Consumer} instance that will receive an instance of the object.

2.4 Project Layout

`/2.0 Conventions/2.4 Project Layout`

Package Structures

3.0 Project Management

`/3.0 Project Management`

Managing Versions with Maven

Use the following Maven commands to management the project dependencies:

Description	Maven Command
Display build plugins with newer versions	mvn versions:display-plugin-updates
Display dependencies with newer versions	mvn versions:display-dependency-updates
Update dependencies to latest release	mvn versions:use-latest-releases

Reference Versions Maven Plugin

Maven Site Reports

Maven site generates the following reports that are used to manage the project dependencies:

Report	Report Location
Project Dependency Management	site/dependency-management.html
Project Plugin Management	site/site/plugin-management.html
Project Build Plugins	site/plugins.html

Tools

Maven 3.8.6 Jave 1.17 C4 Builder

4.0 Dependencies

`/4.0 Dependencies`

4.1 Frameworks

`/4.0 Dependencies/4.1 Frameworks`

3.1.1 Application

`/4.0 Dependencies/4.1 Frameworks/3.1.1 Application`

Dependencies

Spring Boot

The following Spring dependencies are known to be present and used in Admitto Unum projects.

Group	Artifact ID	Description
org.springframework	spring-framework-bom	Import the entire spring boot dependency management

3.1.2 Validation

`/4.0 Dependencies/4.1 Frameworks/3.1.2 Validation`

Validation

Group	Artifact ID	Description	Site
jakarta.validation	jakarta.validation-api	Jakarta Bean Validation Java specification	https://beanvalidation.org/

4.2 Common Reusable Libraries

`/4.0 Dependencies/4.2 Common Reusable Libraries`

Apache Commons

Group	Artfact	Description	
org.apache.commons	commons-text	Text is a library focused on algorithms working on strings	https://commons.apache.org/proper/commons-text/
org.apache.commons	commons-lang3	host of helper utilities for the java.lang API	https://commons.apache.org/proper/commons-lang/

Reference: Apache Commons

4.3 Logging API

`/4.0 Dependencies/4.3 Logging API`

Logging API

Group	Artfact	Description	
ch.qos.logback	logback-core	Java Logging API	https://logback.qos.ch/
org.slf4j	slf4j-api	Simple Logging Facade for Java	https://www.slf4j.org/

4.4 Data Modeling

`/4.0 Dependencies/4.4 Data Modeling`

Data Modeling Dependencies

These dependencies are used to create data classes for models:

Group	Artfact	Description	
org.eclipse.collections	eclipse-collections	Library of collection-management utilities	https://github.com/eclipse/eclipse-collections/blob/master/docs/guide.md
com.fasterxml.jackson.core	jackson-databind	general-purpose data-binding functionality	https://github.com/FasterXML/jackson-databind
com.fasterxml.jackson.dataformat	jackson-dataformat-yaml	Jackson standard text-format dataformat backends	https://github.com/FasterXML/jackson-dataformats-text
com.fasterxml.jackson.datatype	jackson-datatype-jsr310	Support JSR-310 (Java 8 Date & Time API) data types.	https://jcp.org/aboutJava/communityprocess/pfd/jsr310/JSR-310-guide.html
com.fasterxml.jackson.module	jackson-module-parameter-names	Jackson modules needed to	https://github.com/FasterXML/jackson-modules-java8

		support Java 8 features	
org.projectlombok	lombok	Data & Entity Modeling utility	https://projectlombok.org/features/all

4.5 Reflection and Retrospection API

`/4.0 Dependencies/4.5 Reflection and Retrospection API`

Reflection and Retrospection API

Group	Artfact	Description	
org.jooq	joor	Object Oriented Reflection	https://github.com/jOOQ/jOOR
org.reflections	reflections	Java runtime metadata analysis	https://github.com/ronmamo/reflections

4.6 Maven Plugins

`/4.0 Dependencies/4.6 Maven Plugins`

Maven

Build Dependencies

Group	Artfact	Description	
admitto-unum.route66.tools	route66-template-maven-plugin	Build time document templating	
org.apache.maven.plugins	maven-checkstyle-plugin	coding standards	https://checkstyle.sourceforge.io/sun_style.html
org.apache.maven.plugins	maven-compiler-plugin	Java Compiler w/Preview features enabled	
org.apache.maven.plugins	maven-dependency-plugin	Dependency Management & Reporting	
org.apache.maven.plugins	maven-jar-plugin	Provides the capability to build jars	https://maven.apache.org/plugins/maven-jar-plugin/
org.apache.maven.plugins	maven-project-info-reports-plugin	Maven Project Info Reports	https://maven.apache.org/plugins/maven-project-info-reports-plugin/
org.apache.maven.plugins	maven-site-plugin	Generate a site for the project	https://maven.apache.org/plugins/maven-site-plugin/
org.apache.maven.plugins	maven-surefire-plugin	Run and create Unit Test report	https://maven.apache.org/surefire/maven-surefire-plugin/index.html
org.codehaus.mojo	versions-maven-plugin	Manage the versions of artifacts	https://www.mojohaus.org/versions-maven-plugin/
org.owasp	dependency-	detect publicly	https://owasp.org/www-project-dependency-

	check-maven	disclosed vulnerabilities	check/
org.projectlombok	lombok-maven-plugin	Data & Entity Modeling utility	https://projectlombok.org/features/all
net.revelc.code.formatter	formatter-maven-plugin	Reforemat code to coding standards	

Maven

Report Dependencies

Group	Artfact	Description