Development Standards

Appech Inc

1. Development Standards

1.1. Properties

Author	Appech Inc.
Company	
Description	This documentation is designed to explain the development standards for Appech Inc. Development Projects. It is expected that all companies working for Appech Inc. on any project will follow these guidelines for developing quality software.
UML Version	2.x

2. Glossary Terms

2.1. camel cased

Camel casing Capitalizes the first letter of every word in a name with the exception of the first word. totalAmount taxValue ...

2.2. Feature Wrapping

The process of wrapping code for a feature in an If block that checks to ensure that the code is disabled if a feature flag in the configuration is disabled.

2.3. Field

A variable attribute for a class in C# that is not a property.

2.4. Pascal Casing

Pascal Casing joins words together in a single name, and Capitalizes each word in the name. TaxAccount, DayOfWeek

2.5. RPC

RPC is short for Remote Procedure Call. This is a type of API Call that performs an action on a resource. This type of call is not fully restful beyond level 1 due to the fact that there is not an HTTP Verb that describes it's purpose.

POST /api/invoices - would not be a RPC, because it is well established that POST calls are used to create new records.

POST /api/invoices/{id}/calculate-tax-amount - is a RPC. We can tell because in this case the POST Verb is not being used to Create Data.

3. Code Style Rules

3.1. Names

ID: REQ003

The names that are used to identify objects in our code should clearly identify the intended purpose of the object. Variables should be named for what they are: firstCar, lastHouse, total. Methods should be named for what they do: CalculateTax, Dispose, Initialize. Classes should be named for their purpose: EntityExtensions, PersonViewModel, Person.

3.2. Class Name Casing

ID: REQ011

Classes should use Pascal Casing.

3.3. Class Names

ID: REQ006

Class Names should clearly define the purpose for a Class. For instance a class that contains custom TagHelpers for working with decimals should be called DecimalTagHelpers, and a class that provides extensions for a data model should be called DataModelExtensions.

3.4. Constant Casing

ID: REQ009

Constants should be named using All Caps, and words should be separated with an underscore. CITY_VALUE, DEAULT_TAX

3.5. Method Name Casing

ID: REQ012

Methods should always us Pascal Casing, regardless of accessibility.

3.6. Method Names

ID: REQ005

Method names should accurately describe the intended purpose of a method. For instance, one can expect that ToggleActiveStatus would turn off active status when it is on, and Turn on active status when it is off.

3.7. Names for Variables

ID: REQ004

Variable should be given a name that clearly indicates the purpose for the variable. That name should not be contracted, or a shorthand. ExpDate is not valid, but ExpirationDate would be.

3.8. Naming Conventions

ID: REQ001

All variables, definitions, and objects within an Appech project are required to follow the naming conventions outlined in this requirement.

3.9. Parameter Casing

ID: REQ002

Pareameters should be camel cased.

3.10. Private Field Casing

ID: REQ008

Private fields should be Camel Cased with an underscore prefix. _taxAmount, _context

3.11. Property Casing

ID: REQ010

Properties should use Pascal Casing.

3.12. RPC Endpoints

ID: REQ013

When Developing a WebApi using ASP.Net Core, RPC Endpoints should have a named route that explains their purpose. This route name should be hyphen separated. /api/invoices/calculate-tax-rate

calculate-tax-rate is the endpoint in this example. it's clear that this action is performed on the invoice resource object.

3.13. Test Names

ID: REQ007

Test names should describe three things about a test:

1. What is being tested
2. What conditions are being tested
3. What is the expected outcome

This can be done by using the following format for naming a test:

<What><Should><When>
GetTaxTotal_ShouldReturnZero_WhenNoItemsArePresent

4. Code Quality Rules

4.1. SOLID

ID: REQ015

Developers are expected to adhere to the SOLID Design Principals.

4.2. Single Responsibility

ID: REQ016

A class should only have a single responsibility, that is, only changes to one part of the software's specification should be able to affect the specification of the class.

4.3. Classes with Single Responsibility

ID: REQ017

When designing a Class for use in an Appech Project, you should consider what the purpose for that class is. For instance, if you are designing a class for a Data Model, you should avoid adding logic to the class. Logic would extend the original purpose of the class, and you should instead create a second class to operate on the first class with your needed logic.

You should also avoid adding code to a class that does not coincide with the purpose of the class.

4.4. Methods with Single Responsibility

ID: REQ018

Methods should also have a single stated purpose. That purpose can be to orchestrate a series of actions, or to perform an action, but it should never be both. Orchestration methods should have limited logic, and we should never write a unit test for an orchestration method. Action methods should contain your logic, and should be Independent of other methods to allow for easy unit testing.

4.5. Open/Close

ID: REQ019

"Software entities ... should be open for extension, but closed for modification."

Often times when designing software, we find that code requires modification, and the general instinct is to modify the code in place. This should be avoided when possible. If you are working on existing functionality, you should avoid the compulsion to change what might already be in use elsewhere in your code. Instead look for ways to extend the existing code to add your functionality. This can be done with Overrides, and Extension Classes in most cases. Instead of changing existing models, create inherited classes that extend the original. This method of

developing helps to prevent breaking changes that could affect the work of other developers.

5. Agency Development Rules

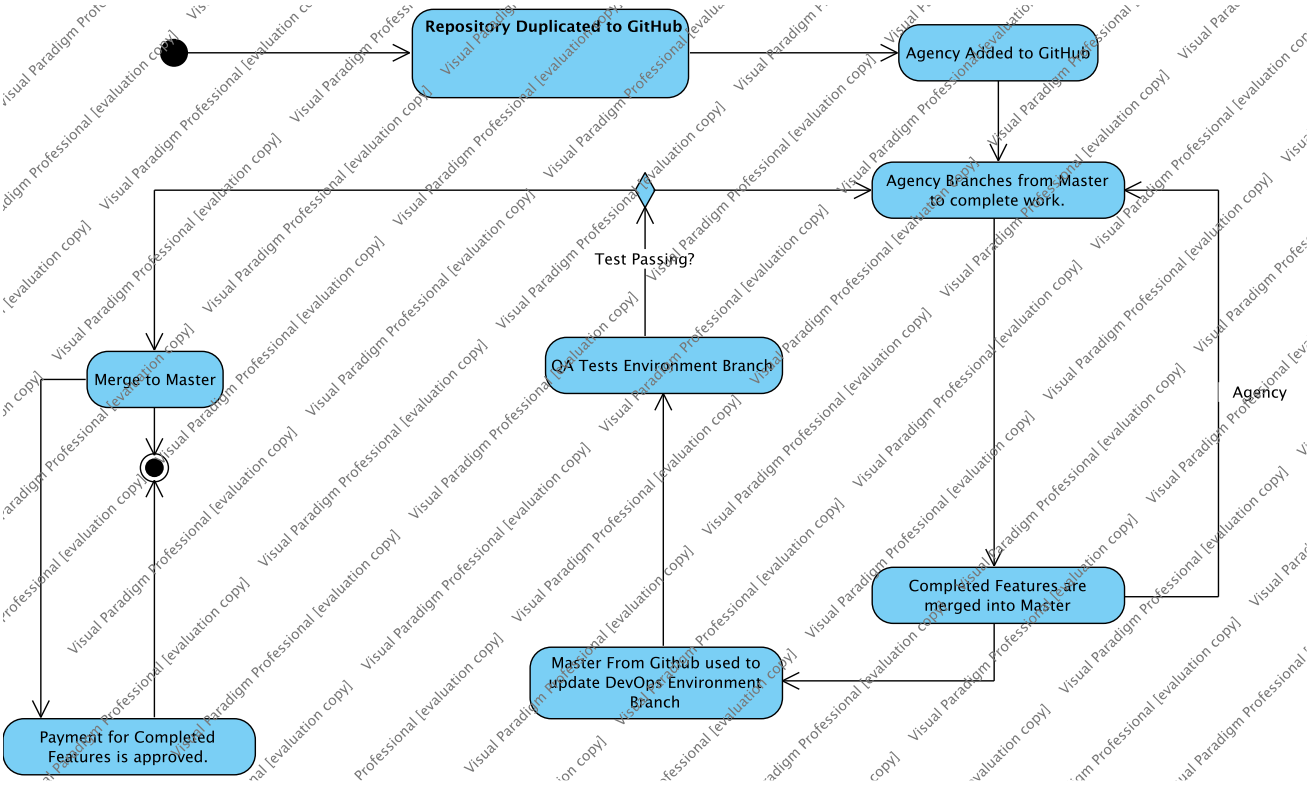
5.1. Agency Development Process

Agency can develop using what ever techniques they choose to develop with. Changes to Master are reviewed daily on the Agency Repository for Standards. When a Feature is reported as completed the code is moved back to the Internal Repository for review.

To be paid for a completed feature the following must occur.

1. Code Quality Inspection must pass for all code in master.
2. When deployed to a testing environment the code must pass QA inspection, and features must function as expected.
3. Prior to merging to Internal Master all included functionality must be completed, or disabled with Feature Wrapping.

5.2. Agency Development Process Activity Diagram



5.2.1. Agency Added to GitHub

5.2.2. Agency Branches from Master to complete work.

5.2.3. Completed Features are merged into Master

5.2.4. Master From Github used to update DevOps Environment Branch

5.2.5. Merge to Master

5.2.6. Payment for Completed Features is approved.

5.2.7. QA Tests Environment Branch

5.2.8. Repository Duplicated to GitHub

5.2.9. Test Passing?

5.2.10. unnamed

5.2.11. unnamed